AFRL-IF-RS-TR-2002-6
Final Technical Report
February 2002

FORMAL METHODS FRAMEWORK

WetStone Technologies

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-6 has been reviewed and is approved for publication.

APPROVED:

JAMES L. SIDORAN
Project Engineer

FOR THE DIRECTOR:

WARREN H. DEBANY, Technical Advisor
Information Grid Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFGB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FEBRUARY 2002	Final Jun 99 - Jun 00

4. TITLE AND SUBTITLE
FORMAL METHODS FRAMEWORK

5. FUNDING NUMBERS
C - F30602-99-C-0166
PE - N/A
PR - 1065
TA - 09
WU - P2

6. AUTHOR(S)
Chester Hosmer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
WetStone Technologies
17 Main Street, Suite 237
Cortland New York 13045

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFGB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2002-6

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: James L. Sidoran/IFGB/(315) 330-3174

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
This contract final technical report documents the Formal Methods Framework (FMF) project results. This project considers the impact of formal methods on industries such as high assurance and telecommunications software and proposes a framework within which formal method can be used more effectively to produce reliable and correct software. A FMF is developed an populated, and areas of future improvement such as extensibility, scalability and range of applications are identified.

14. SUBJECT TERMS
Formal Methods, Classification, Telecommunications

15. NUMBER OF PAGES
106

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

Table of Contents

Table of Contents (Continued)

1 Period of Performance

This report reflects performance from 5/26/99 through 10/26/99.

2 Detailed Program Schedule

The following represents the schedule for this project:

Progress	Months					
	Jun	Jul	Aug	Sep	Oct	Nov
Perform initial research & project setup	4					
Form Collaborations / Research Tools / Distribute questionnaires						
Examine tools for possible framework properties						
Identify framework properties						
Prepare Final Report						

3 Background

Our survey of the current practices in formal methods in academia and industry [Barj98] indicates that formal methods (FM) are a promising technology that is eliciting more and more industrial interest. Major issues in software and hardware industry are complexity and size, and current practices such as simulation cannot perform to the desired level of satisfaction anymore.

In the hardware industry, formal tools are popular and adopted in standard engineering practice. Many tool vendors such as Crysallis or Synopsis make formal tools and/or integrate formal tools in their commercial CAD toolkits. For Example, Cadence is currently producing a "Verification Cockpit" toolset. Incentives are: high cost of design errors, standard notation (VHDL/Verilog), and use of standard tools. Formal methods replace simulation, with the prevalent use of model checking to reveal errors.

In the commercial software industry, there is none or very little use of formal methods. The barriers include: product patches are distributed electronically, software is written in many languages, there is very little use of any tools, and software engineering is not a discipline based on formalism and mathematics such is digital design.

High assurance and telecommunications software industry use formal tools to some degree, with their use increasing. Telecommunications industry is driven by (often international) standards

compliance and need for test–case derivation. Information security industries, such as electronic commerce and banking, network security, and military applications are motivated by the virtue of information as commodity, with tangible material and strategic cost. Safety critical applications applications, such as avionics, medicine, railroads, and nuclear power applications are motivated by having human lives at stake.

Most formal methods practitioners agree that many additional steps are needed to take formal methods from research to industrial practice. Most commonly mentioned features include: infrastructure, such as robust and supported tools, easy to use, with verified libraries; publicizing success stories; and user education.

Some preliminary work can be done in order to make formal methods more approachable to users. IEEE Formal Methods Planning Group met in an open meeting in November 1998 at SRI International, Menlo Park, CA, to discuss what steps, if any, can be taken towards standardization of formal methods. The consensus was that standardization is premature, and that it would be necessary to collect information on the existing formal tools and somehow classify them, and collect and standardize formal methods terminology. The project described in this report addresses those concerns.

We attended World Congress on Formal Methods, which was attended by about 500 formal methods specialists from all over the world. During 1.5 hr meeting called IEEE Formal Methods Planning Group Birds-of-a feather meeting, formal methods experts discussed what needs to be done in formal methods, using our work on tool classification and taxonomy. The resulting recommendations are included in the conclusion.

A long term goal of WetStone Technologies, Inc. is to produce a robust, industrially usable Formal Methods Framework (FMF) that is populated by several formal methods and tools. This framework must be extensible, scaleable, and general enough to address a range of application problems but specific enough to address desired application domains. An undertaking of this size would require partnership between several teams with different expertise and several years of work. In this effort, we are taking the first step by outlining the preliminary work necessary to pave the way for the creation of the fully developed Formal Methods Framework.

3.1 Existing Tool Classification and Terminology Documents

Some documents and databases which outline formal methods terminology, tools used and experience reports already exist. We will not discuss databases of links to various tool pages, such as [BoweWWW], but rather databases which attempted to classify tools based on some predetermined criteria.

Formal Methods Europe (FME) is "an organization supported by the Commission of the European Union, with the mission of promoting and supporting the industrial use of formal methods for computer systems development." FME organizes seminars and a yearly international symposium, and produces a newsletter. FME's web page [FME] contains some case studies, formal methods database, and a tools database. The case studies database seems not to be up to date, and the tools database seems not to be up to date, with the latest additions in 1997, although the web page claims 6-month updates. The tools database contains about 60 international tools and is, in our opinion, suitable for a quick overview of tools. The tools are classified by the following categories:

- Tool name
- Usage and applicability
- Languages supported
- List of applications (if available)
- Functionality: yes/no answers to the following:
    - Syntax checking
    - Static semantics
    - Animation.execution
    - GUI
    - Pretty-print
    - Typechecking
    - Proof support
    - Refinement
    - Test-case generation
- Environment, number of installations, last update
- Contact
- Availability
- Description.

European Workshop on Industrial Computer Systems (EWICS) Formal Methods subgroup produced documents that contain some formal methods terminology, formal methods database, and a classification of methods by their theoretical basis [EWICS98]. EWICS formal methods database is relatively current (dated June 1998) but it contains only CCS, COLD, OBJ, SAGA-LUSTRE, Z, RAISE, B and VSE formal methods, and it focuses more on methods than on tools. The methods are classified as:

- Formal method name
- Summary
- Applications
- Properties
- Relation to other formal methods
- Theoretical basis
- Tools
- Appraisal:

    - Maturity
    - Availability
    - Strength
    - Industrial experience
    - Tool availability
- Application and experience matrix
- Tools matrix
- Bibliography

Craigen, Gerhart and Ralston have published "An International Survey of Industrial Applications of Formal Methods" [CrGeRa93], which contains much valuable information but is dated as 1993. [ClWi96] paper has some experience reports as well and is more recent. Experience reports need to be kept in an up-to-date database available on the web.

Some definitions of formal methods terms are published in the following reports and databases, such as: NASA's Formal Methods Guidebooks [NASA97, NASA98]; EWICS' Guide [EWICS98]; Laprie's report "Dependability: Basic Concepts and Terminology" [Lapr]; "Dictionary of Algorithms, Data Structures and Problems" [Black99], compiled by Paul Black for CRC Dictionary of Computer Science, Engineering and Technology; and Rushby's technical report on "Formal Methods and Their Role in the Certification of Critical Systems" [Rush93, Rush95]. Various formal methods terminology is scattered throughout the published literature, such as [ClWi96]. Effort is needed to collect the terminology as is used today and converge it into a common terminology, i.e. formal methods "lingua franca."

4 Project Activity

The immediate goals for this project were to:
1. Collect terminology and develop a taxonomy of terms used in formal methods
2. Classify a subset of formal tools.

Our intent is to contribute to a more widespread use of formal methods by making formal methods more accessible and understandable to potential users, including industrially-oriented users new to formal methods. We developed a questionnaire that should help potential and new users assess what tools are available for their needs. The questionnaire was used to collect information on selected tools, and develop classification and taxonomy based on the collected information.

We have presented this work at World Congress on Formal Methods (FM'99) during IEEE Formal Methods Planning Group Birds-of-a-Feather meeting. Discussion ensued that points in the direction of future work and confirms the orientation towards industrial practitioners. The main points of the discussion are outlined in the "Conclusions" section.

4.1 Classification of Formal Tools

We compiled information on the best-known and widely used formal tools, with emphasis on tools aimed at industrial practitioners without extensive formal methods expertise. The tools are:
1. Theorem provers: PVS, ACL2, HOL, Larch LP tool, Z/EVES;
2. Model checkers: SMV, SPIN, Murphi, Concurrency Factory;
3. Other tools: NRL Protocol Analyzer, SCR*, Tatami.

We have devised a questionnaire to aid in collecting and classifying this information. There are many criteria for classifying the tools, based on the intended use of the tool survey. Possible audiences include tool developers, industry/users, and academia/researchers. We have assumed that the tool survey will be used by users new to the tools to aid them in selecting appropriate tools. We envisioned users who are interested in practical application of the tools and possibly do not have extensive background in formal methods. We chose the main categories for classifying the tools to be:

1. general description of the tool;
2. tool implementation (such as what language the tool is implemented in, is the tool extensible);
3. tool features and utilities (such as validated libraries, GUI, typechecking, prettyprinting, editing);
4. tool input and output;
5. tool applications (such as application domains, levels of abstraction;
6. resources required to run the tool (such as licensing, platform, operating system);
7. resources available (such as manuals, courses, contacts);
8. more specific detailed questions pertaining specifically to model checkers and theorem provers; and
9. open-ended questions for quick assessment of tools' strengths and weaknesses, and a list of case studies and experience reports.

We have designed the basic questionnaire and revised it based on the feedback from the Engineering Consortium, various verification mailing lists, and SRI CSL. We also modified the questionnaire for on-line filling.

For each tool, we filled the questionnaire as a "new" user, i.e. we have studied the readily available literature about the tool as if we are evaluating it for potential use. Questionnaires were then distributed to tool makers and user mailing lists for feedback. Returned questionnaires were edited for consistency between various responses. All questionnaires came back with feedback except for Larch LP, Tatami, and NRL Protocol Analyzer. (According to Jeannette Wing at FM'99, use of Larch language and tools is on the sharp decline and that might explain lack of interest in participating in this survey.)

The questionnaires are in the Appendices. We posted the questionnaires on WetStone's web page, as http://www.wetstonetech.com/fm_quest.html, and requested that it be linked to various formal methods web repositories, such as Engineering Consortium page and World Wide Web Virtual Library on Formal Methods. The questionnaires were presented at the World Congress on Formal Methods (FM'99).

4.2 Taxonomy of Formal Terms

We have examined [NASA97, NASA98] FM guidebooks, various papers on formal methods including [ClWi96] and many others, various technical reports such as [Rushby95] and combined existing definitions into a formal methods terminology. The taxonomy is application-domain independent. It is intended to satisfy a wide range of users, including practicing engineers who might not be fully trained in mathematical logic. For a more theoretical treatment of technical details involved in formal methods, a reader is referred to textbooks on logic and theoretical studies of languages such as [EWICS98]. The taxonomy is presented in the Appendix, and posted on WetStone's web page at http://www.wetstonetech.com/fm_quest.html.

5 Conclusions and Future Work

The following work is needed to move formal methods into a more mainstream practice:

1. A common terminology. Various differing definitions need to be converged into a common terminology to be accepted as the "lingua franca" of formal methods, and potentially standardized.
2. Common APIs and exchange formats for tool interoperability, potentially to be standardized.
3. Classification of formal methods, based on their language, method, and tool, as well as the relationship between them. Ideally, also include classification based on application domains.
4. Guidelines for using formal methods in industrial practice, including the following:
   a. Overview of the state-of-the-art in formal methods practice.
   b. A classification of tools, containing short overview and description of each tool, time-stamped and indicating if the tool is industrial strength or research prototype.
   c. A questionnaire which users can use to guide them in selecting tools.
   d. Experiences database, organized by application type and industry area; or, for each tool, what types of problems it was used for.
   e. Examples done in each tool, using similar problems as benchmarks.
   f. A catalogue of formal methods courses, training, books, and other educational resources.
   g. "Method behind the method" for tools, i.e. how can each method/tool be used and/or what is its theoretical basis for implementation.
   h. A bibliography of links to the above information, to be posted on a web site.
5. Developed "infrastructure," such as verified libraries and transition from research prototype tools into industrial strength tools.
6. Integration of tools into toolkits, and integration of tools into industrial process flow.

This project has accomplished items 3.b and 3.c, and produced the first draft of item 1. [Barj98] addressed item 3.a, but the overview needs to be updated yearly. The future work would be to address the remaining items.

5.1 Long-term Future Work: Formal Methods Framework

In order to integrate tools into a framework, items 2 and 3.g must be completed first. Our perspective and long-term goal is to identify robust tools that can be integrated together in a formal toolkit or added to existing toolkits. In order to accomplish that, we need to:

- Identify application or class of problems. Possible choices at this moment seem to be:
  - hardware/software co-design
  - information/networking security
  - system-level design.
- Identify collaborators. Tool integration can be achieved only with the assistance of tool makers. We need to identify collaborators that can bridge the gap between research and industrial practice.

Potential collaborators include: Derivation Systems (contact Dr. Bhaskar Bose); Dr. Perry Alexander (U of Cincinnati); SLDL project (contact Dave Barton, Intermetric Inc.) and the tool integration group at Ptolemy Project (contact Dr. Edward Lee, U of California at Berkeley).

Derivation Systems company is dedicated to making industrial formal methods products. Employees are Ph.D.-level trained in formal methods. Therefore, this company provides expertise in commercial applications of formal methods research. Furthermore, the company sells formal hardware tools, and recently has acquired software expertise in formal network assurance for secure Java applets.

SLDL (System-Level Design Language) project is an ongoing, industry-driven effort to develop a language and its tool support for describing systems-on-silicon. SLDL is intended for use by electrical engineers designing microsystems with embedded software. SLDL is of interest to our project because of its plug-and-play architecture. SLDL framework will include bridging the semantics of several existing domain-specific system languages (e.g. Esterel, SDL, and C++).

Dr. Perry Alexander has been involved in several projects that bridge the gaps between formal methods research and industrial practice, hardware and software. For example, CEENS project (sponsored by Air Force), SLDL project, and HEPE project (sponsored by DARPA/ITO). CEENS project had the goal to develop methodology and tools necessary to support board and module level of electronic integration and develop a commercial products. The project involved collaboration between Dr. Alexander and commercial companies TRW, Motorola, and Mentor Graphics, and included industry review board. HEPE project deals with high assurance heterogeneous network assurance prediction, and thus provides with software experience.

Some of the tools we have examined already integrate with other tools, for example PVS integrates with SCR*, which integrates with SPIN. The trend is between integration between theorem provers and model checkers, such as in PVS and SMV.

There are many ways to integrate tools. Ideal toolkit would consist of a "stack" of tools that can address various levels of abstraction to aid in development by transformation. Ideally, tools would be able to share common data. In practice, tools have been integrated based on shared APIs and sockets (such as in Z/EVES); or common meta-language (such as in UniForm and Express IT toolkits, and many commercial non-formal CAD toolkits) or some logic as the logical framework (such in Maude). We envision a formal methods framework that integrates several tools in an open environment. The tools should be either extensible or with provided API, and contain many "convenience" non-formal tools, such as typecheckers, editors and prettyprinters, as well as validated libraries. What is needed is more validated libraries for theorem proving, and macros of temporal logic formulas for model checking. Our goal would be to integrate theorem proving and model checking in an efficient way.

Tools which look promising for such integration are PVS, SCR* and SPIN, since they have already begun their integration. For example, PVS already contains a model checker, but it would be more efficient from a user's point of view to not have to learn another tool, e.g. an experienced SPIN user should be able to supply input to PVS and vice versa.

An outcome of this work would be to produce guidelines on how to express various properties in various tools, which is one of the features states as "needed" by the formal methods community.

An advantage of combining tools such as PVS and SPIN is the possible extent of cooperation from tool vendors and users. PVS distribution does not contain source code, so it is not user extensible, but PVS is a product of a commercial company and there are human resources available to extend the tool, even though PVS itself is free. SPIN source code is freely available and often extended by users. The combination of the two could result in a tool suite that is free and capable of integrating model checking and theorem proving in an effective way.

5.2 Travel

Date	Destination	Purpose of Trip

5.3 References

[ACL2] Applicative Common Lisp (ACL2) home page, http://www.cs.utexas.edu/users/moore/acl2/index.html.

[Barj98] Milica Barjaktarovic, "The State-of-the-art in Formal Methods," AFOSR Summer Research technical report for Rome Research Site, AFRL/IFGB. http://www.wetstonetech.com/fm_quest.html.

[Blac99] Paul Black, ed., "Dictionary of Algorithms, Data Structures, and Problems," compiled originally for the CRC Dictionary of Computer Science, Engineering and Technology. http://hissa.ncsl.nist.gov/~black/DADS.

[BoweWWW] Jonathan Bowen (webmaster): "WWW Virtual Library on Formal Methods", http://www.comlab.ox.ac.uk/archive/formal-methods/, links to individual tools' pages.

[ClWi96] Edmund M. Clarke, Jeannette M. Wing, at. al. "Formal Methods: State of the Art and Future Directions." *ACM Computing Surveys*, 28(4):626-643.

[EWICS98] European Workshop on Industrial Computer Systems, Technical Committee 7 (Safety, Reliability, Security), Formal Methods subgroup, "Guidance on the Use of Formal Methods in the Development of High Integrity Industrial Computer Systems." Parts I, II, working paper 4001, June 1998. http://www.ewics.org.

[EWICS98'] European Workshop on Industrial Computer Systems, Technical Committee 7 (Safety, Reliability, Security), Formal Methods subgroup, "Guidance on the Use of Formal Methods in the Development of High Integrity Industrial Computer Systems." Part III, "A Directory of Formal Methods," working paper 4002, June 1998. http://www.ewics.org.

[Factory] Concurrency Factory home page, http://www.cs.sunysb.edu/~concurr

[FME] Formal Methods Europe home page, http://www.cs.tcd.ie/FME/, or http://www.fme-nl.org.

[HOL90] HOL home pages, http://www.comlab.ox.ac.uk/archive/formal-methods/hol.html

[Lapr] Jean-Claude Laprie, "Dependability: Basic Concepts and Terminology." Laboratory for Analysis and Architecture of Systems (LAAS) - CNRS, LAAS report No92043. http://www.laas.fr.

[Murphi] Murφ homepage, http://sprout.stanford.edu/dill/murphi.html.

[NASA98] "Formal Methods Specification and Verification Guidebook for Software and Computer Systems, Volume I: Planning and Technology Insertion" [NASA/TP-98-208193], 1998. http://eis.jpl.nasa.gov/quality/Formal_Methods/

[NASA97] "Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems, Volume II: A Practitioner's Companion" [NASA-GB-001-97], 1997. http://eis.jpl.nasa.gov/quality/Formal_Methods/

[NRL] NRL home page, http://www.itd.nrl.navy.mil/ITD/5540/projects/crypto.html

[PaulWWW] Larry Paulson (webmaster): http://www.cl.cam.ac.uk/users/lcp/hotlist#Systems

[TalcWWW] Carolyn Talcott (webmaster): http://www_formalstanford.edu/clt/ARS/ars-db.html

[PVS] PVS home page, http://pvs.csl.sri.com

[Rush93] John Rushby, "Formal Methods and Their Role in the Certification of Critical Systems", SRI International Technical Report CSL-93-7, March 1993. http://csl.sri.com/csl-93-7.html.

[Rush95] John Rushby, "Formal Methods and Their Role in the Certification of Critical Systems", SRI International Technical Report CSL-95-1, March 1995. http://csl.sri.com/csl-95-1.html.

[SCR] SCR home page, http://www.chacs.itd.nrl.navy.mil/SCR

[SMV] SMV home page, http://www-cad.eecs.berkeley.edu/~kenmcmil

[Spin] Spin home page, http://netlib.bell-labs.com/netlib/spin/whatispin.html

[Z/EVES] Z/EVES home page, http://www.ora.on.ca/z-eves.

Appendix A – Questionnaires

ACL2

```
**********************************************************************
****************************** ACL2 ***********************************
***************************Sep. 1999***********************************

For this particular tool, please answer the following questions grouped
based on: general description of the tool, tool implementation, tool
features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

o Rough classification:
      ____ model checker
      ____ theorem prover
      ____ mechanized proof assistant
      _X__ other: _integrated toolkit: logic, mechanized proof assistant,
                   executable model environment.
o Application domain(s) or class(es) of problems originally intended.
      Formal verification o digital systems.
      Building executable models that can be run and/or symbolically
executed.
o Intended audience.
      Engineers and mathematicians working on industrial-strength
applications.
      More generally, anyone wanting to reason about formal models.
o Language(s) and/or technique(s) that the tool is based on.
      ACL2 logic (a subset of first-order applicative Common Lisp, i.e.
      excluding non-applicative aspects such as higher-order functions,
      circular structures, and Common Lisp Object System).
o Reasoning mechanisms used for the tool.
      Mathematical induction, rewriting, decision procedures (equality, BDDs,
      linear arithmetic), heuristics
o Comparable languages/tools.
      HOL, PVS, (Pc-)Nqthm.
      ACL2 is industrial-strength successor of Boyer-Moore theorem prover
      Nqthm).

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation.
      Applicative Common Lisp (Allegro, GCL, Lispworks, Lucid, MCL).
o How extensible and/or customizable is the tool.
      _X__ source code given
      _X__ tool implemented in a public-domain language
      _X__ other: _users post libraries___________
                  Features enabling modification include
                  extensive comments in sources and applicative
                  coding style (e.g., no global variables).
```

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):

____ GUI

_X__ Library of standard types, functions, and other constructions

_X__ the library is validated

The extent of the library is (speaking from the point of view of a potential user):

___ not very comprehensive

_X_ reasonably comprehensive

___ quite comprehensive

_X__ Editing and document preparation tools

__GNU Emacs______________

__ACL2 event files can be published in LaTex, HTML, Scribe, or ASCII text. Formatting is user-extensible.____________

____ Cross-referencing

____ Browsing

____ Requirements tracing

_X__ Incremental development across multiple sessions

____ Change control and version management

_X__ Consistency checking (via the "encapsulate" form)

_X__ Completeness checking (in the sense that theorems can be proved)

_X__ Other:

_infix interface to ACL2, to ease familiarizing with ACL2 for those not familiar with Lisp prefix syntax.____________

____

____

o How interactive/mechanized/automated is the tool.

_X__ fully automated (model execution)

_X__ user guided (theorem prover)

____ other: ____________

4. TOOL INPUT AND OUTPUT

o Tool supports these models:

_X__ synchronous

_X__ asynchronous

_X__ mixed

o Input to the tool.

Model in ACL2 and proof hints.

o Output from the tool.

Proof results.

Execution results.

o The language used for input to the tool has (check all that apply):

_X__ formal semantics

_X__ modern programming language constructs (e.g. if-else):

____

____ strong typing
_X__ modularity
_X__ hierarchical design
_X__ parameterization
(in the sense that functions can be parameterized)
____ communication between processes
____ buffered
_X__ built-in model of computation
____ other: ____________

5. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):
_X__ requirements
_X__ design specification
_X__ implementation
_x__ test derivation
(not part of the system, but conveniently user-implementable)
_X__ RTL
_X__ netlists
____ transistor level
_X__ other: In principle, any level can be addressed, but some levels would require more work than others.

o Has the tool been integrated with other tools?
____ no
____ yes
with ____________________
with ____________________
_X__ do not know
_Many loose integration, via translators into ACL2,
_but no tight integration known to tool makers.

6. RESOURCES

o Resource requirements for the tool:
UNIX version __Sun OS, Linux________
Windows version __________
Mac version _X____
Memory: __at least 16MB, preferably at least 64MB__

o Cost, rights and restrictions:
____ free, no license
_X__ free, license required
(GNU General Public License)
___ for educational and research use only
____ nominal distribution charge
____ fee for underlying tool(s)
____ flat license fee
____ per user license fee
____ royalties per use

____ other: ____________

o User background prerequisites (check all that apply):

- _X__ BS degree
- ____ MS degree
- ____ Ph.D. degree
- _X__ knowledge of logic
  - _X__ first-order
  - ____ high order
- ____ familiarity with a high-level programming language
- ____ familiarity with process algebra
- ____ familiarity with temporal logic
- _X__ other: __minimal familiarity with Common Lisp__________

____

____

o User's learning curve, if all prerequisites are met:

- ____ one month
- ____ two months
- _X__ less than six months
- ____ other:
  - ______ months

o Tool support

- _X__ upgrades/maintenance
  - Last version produced at this date: _ACL2 v.2.4, 1999__
- _X__ manual
  - _X__ on the web
- _X__ training
  - (tutorials on the web)
- _X__ listserv
- ____ mailing list
- _X__ dedicated conference(s)/workshop(s)
  - (One held in March 1999; next is anticipated in Oct. 2000)
- ____ human "help line"
- _x__ book(s)
  - (To appear in 2000).
- _X__ journal/conference publications
- _X__ other: _bug reports to acl2@lists.cc.utexas.edu__
  - _libraries, hypercard on the web____________

o Current contact.

http://www.cs.utexas.edu/users/moore/acl2/index.html
acl2@lists.cc.utexas.edu (subscribe to acl2-request@lists.cc.utexas.edu)

7. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

o Verification mechanism(s) (check all that apply):

- ____ equivalence
- ____ modal logic

____ temporal logic
____ system or process invariants
____ built-in support for checking for:
___ deadlock
___ livelock
___ other: ____________
____
____ other: ____________
____

o Tool supports (check all that apply):
____ optimization and state reduction mechanism using ________________________
____ simulator:
____ interactive
____ random
____ feedback on in what state verification failed
___ trace leading to the state

8. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

o Degree of proof mechanization.
____ fully mechanized
_X__ partially mechanized

o Support for developing and viewing the proof.
Prover gives output showing the progress of the proof that users typically inspect in order to develop appropriate lemmas (rules) to assist in subsequent attempts. An interactive loop allows finer control of the proof process, as does a tool for monitoring the rewriter. "Proof trees" provide a sort of outline mode for the proof that can ease browsing.

o Presentation of proof to the user.
The proof is presented as formulas that the prover is attempting to reduce to "true".

o Tool supports (check all that apply):
_X__ automated support for arithmetic reasoning
_X__ automated support for efficient handling of large propositional expressions
_X__ automated support for rewriting
_X__ possible to use lemmas before they are proved.
_X__ possible to state and use axioms without having to prove them.
_X__ new definitions can be introduced and existing definitions modified during proof
(at least, if "during proof" is interpreted as "during the proof effort" then this is done all the time)
____ facilities for editing proofs
_X__ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified
Caveat to the above: Some of the basic foundations are collapsed, e.g., as "trivial observations"
_X__ reasonably easy to reverify a theorem after slight changes to the specification

9. OPEN-ENDED QUESTIONS

o Strengths of this tool.
Industrial-strength tool.
Built and based on a programming language, so models can be symbolically executed, run, and theorem-proved.
State-of-the-art heuristics and efficiency for inductive theorem proving.

o Limitations of this tool.
Reasoning directly about quantified notions can be very awkward.
Learning curve.

o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.
Digital systems verification.
Bridging the gap between current practice (simulation) to the goal practice (formal verification) using symbolic execution, or less ambitiously, by providing a formal language for reasonably efficient simulation.

o Applications that the tool was used for - case studies, examples, success stories.
See http://www.cs.utexas.edu/users/moore/publications/acl2-papers.html.
Examples:
industrial microprocessor AMD5K86 and K7 floating-point verification, Motorola CAP DSP design.
Verification of COBOL Year 2000 conversion rules.

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1.
http://eis.jpl.nasa.gov/quality/Formal_Methods/

HOL

```
*********************************************************************
*****************************     HOL     *****************************
*******************************Sep. 1999****************************
```

For this particular tool, please answer the following questions grouped based on: general description of the tool, tool implementation, tool features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

o Rough classification:

____ model checker
____ theorem prover
_X__ mechanized proof assistant
____ other: ________________

o Application domain(s) or class(es) of problems originally intended.

General - from formalizing pure mathematics to verification of industrial hardware.
Has been used for hardware and software verification.

o Intended audience.

General.

o Language(s) and/or technique(s) that the tool is based on.

Higher-order logic interfaced to Standard ML as the meta language.

o Reasoning mechanisms used for the tool.

Higher order logic, using predicate calculus with terms from the typed lambda calculus (i.e. simple type theory).

o Comparable languages/tools.

ACL2, Eves, Isabelle, Nqthm, LAMBDA, LP, Nuprl, PVS
ProofProver (commercial implementation of HOL used fo reasoning about Z specifications)

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation.

Standard ML (Moscow ML for HOL98, New Jersey ML for HOL90).
A non-standard ML for HOL88.

o How extensible and/or customizable is the tool.

_X__ source code given
_X__ tool implemented in a public-domain language
____ not extensible by user
____ other: ____________
____

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):

_X__ GUI
(as a downloadable extension to HOL)
_X__ Library of standard types, functions, and other constructions
_X__ the library is validated

The extent of the library is (speaking from the point of view of a potential user):
___ not very comprehensive
___ reasonably comprehensive
_X_ quite comprehensive

_X__ Editing and document preparation tools
_emacs interface (as a downloadable extension)______________
____

____ Cross-referencing
____ Browsing
____ Requirements tracing
____ Incremental development across multiple sessions
____ Change control and version management
____ Consistency checking
____ Completeness checking
____ Other:
____
____
____
____
____
____

o How interactive/mechanized/automated is the tool.
____ fully automated
_X__ user guided
____ other: ___________

4. TOOL INPUT AND OUTPUT

o Input to the tool.
Higher-order logic proof description.

o Output from the tool.
Proof goals proved or not.

o The language used for input to the tool has (check all that apply):
_X__ formal semantics
_X__ modern programming language constructs (e.g if-else):
____
____
____
_X__ strong typing
_X__ modularity
_X__ hierarchical design
_X__ parameterization
_X__ built-in model of computation
____ other: ___________
____
____

5. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):
_X__ requirements
_X__ design specification

_X__ implementation
____ test derivation
_X__ RTL
_X__ netlists
_X__ transistor level
_X__ other: _mathematics__________
(in principle, every level can be addressed, but lower levels require more work)

o Has the tool been integrated with other tools?
____ no
_X__ yes
with _Isabelle____________________
with _ProofProver___________________
with _CHOL, non-specialist user interface to HOL__
with ____________________
____ do not know
Note: Many extensions and interfaces, such as GUI, Emacs.
Many embedded languages, such as Z, CCS.

6. RESOURCES

o Resource requirements for the tool:
UNIX version ___precompiled binaries for Sun3, Sun4, MIPS, Alpha__
Windows version __________
Mac version ___________
Memory: ____________

o Cost, rights and restrictions:
_X__ free, no license
____ free, license required
____ for educational and research use only
____ nominal distribution charge
____ fee for underlying tool(s)
____ flat license fee
____ per user license fee
____ royalties per use
____ other: ____________

o User background prerequisites (check all that apply):
____ BS degree
_X__ MS degree
_X__ PhD degree
_X__ knowledge of logic
____ first-order
_X__ high order
____ familiarity with a high-level programming language
____ familiarity with process algebra
____ familiarity with temporal logic
____ other: ____________
____
____

o User's learning curve, if all prerequisites are met:
____ one month
____ two months
____ less than six months
_X__ other

___6__ months

o Tool support

_X__ upgrades/maintenance
  Last version produced at this date: __HOL98____

_X__ manual
  _X__ on the web

_X__ training
  (courses at various locations, lectures and tutorials on the web)

_X__ listserv

_X__ mailing list

_X__ conference(s)/workshop(s)
  (annual international intercontinental conference TPHOL)

____ human

_X__ book(s)

_X__ journal publications

_X__ other: __web pages with code depositories and ftp/faq archive__________
  __HOL2000 initiative, to design next generation HOL-like provers_____
  __special journal issues related to HOL
  __user meetings
  __very extensive documentation (tutorial, description, manual, manual for each supported library, primer for beginners, notes, user manuals, applications)
  __bug/problem reports: hol-supprt@cl.cam.ac.uk

o Current contact.
  http://www.cl.cam.ac.uk/Research/HVG/HOL
  info-hol@lal.cs.byu.edu (subscribe at info-hol-request@lal.cs.buy.edu)

7. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

o Verification mechanism(s) (check all that apply):

____ equivalence

____ modal logic

____ temporal logic

____ system or process invariants

____ other: ____________

o Tool supports (check all that apply):

____ optimization and state reduction mechanism(s)
  using _________________________

____ simulator
  ____ interactive
  ____ random

____ feedback on in what state verification failed
  ___ trace leading to the state

____ built-in support for checking for:
  ___ deadlock
  ___ livelock
  ___ boolean propositions
  ___ other: ____________
  ____

8. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

o Degree of proof mechanization.

____ fully mechanized

_X__ partially mechanized

o Support for developing and viewing the proof.

o Presentation of proof to the user (e.g., user input or canonical expressions,
with or without quantifiers).

o Tool supports (check all that apply):

____ automated support for arithmetic reasoning

____ automated support for efficient handling of large propositional expressions

_X__ automated support for rewriting

____ possible to use lemmas before they are proved.

____ possible to state and use axioms without having to prove them.

_X__ new definitions can be introduced and existing definitions modified during proof

____ facilities for editing proofs

_X__ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified

_X__ reasonably easy to reverify a theorem after slight changes to the specification

9. OPEN-ENDED QUESTIONS

o Strengths of this tool.

Powerful proof mechanism for formal verification, induction, infinite data sets. Active and large established user group.

o Limitations of this tool.

Difficult to specify control sequences, takes a long time to learn. Less payoff for lower levels of abstraction.

o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.

Verification of problems containing extensive data path.

o Applications that the tool was used for - case studies, examples, success stories.

Some are posted http://www.dcs.glasgow.ac.uk/~tfm/hol-bib.html
Examples: embedding of various languages (e.g. Z, CCS, hardware languages); security; distributed systems; protocols; hardware; networking elements; compiler verification; real-time systems; reactive systems.

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1. http://eis.jpl.nasa.gov/quality/Formal_Methods/

Larch Prover (LP)

```
**************************************************************************
***************************** Larch Prover (LP)  **********************
********************************** Sep. 1999**************************
```

For this particular tool, please answer the following questions grouped based on: general description of the tool, tool implementation, tool features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

```
o Rough classification:
      ____ model checker
      ____ theorem prover
      _____ mechanized proof assistant
      __X__ other:        __integrated suite of tools:
                            LP mechanized proof assistant, LSL checker and
                            LCLint C program checker.
```

o Application domain(s) or class(es) of problems originally intended.
Software design and verification. Concurrent algorithms in hardware and software. Circuits.
Intended to assist users in finding and correcting flaws in conjectures that need to be proven.

o Intended audience.
Programmers, designers.

o Language(s) and/or technique(s) that the tool is based on.
Multi-sorted first order logic. User specifies axiomative theories to be proved.

Note: each Larch specification contains two components: one written in a Larch Interface Language, which is designed for a specific programming language; and another written in Larch Shared language (LSL), which is independent of any programming language. Larch Interface Languages exists for C (LCL), Ada, Modula-3, VHDL, and others.

LSL tool checks for syntax and type errors in LSL specifications, and can translate it into input files. for LP.

LCLint tool statically checks C programs, including common lint checks such as type inconsistencies, ignored return values, likely infinite loops, as well as assertions about assumptions in desired places in the C code ad errors in dynamic memory management.

o Reasoning mechanisms used for the tool.
Theorem proving, including forward and backward inference, equational term-rewriting, induction rules.

o Comparable languages/tools.
HOL, PVS.

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation.

o How extensible and/or customizable is the tool.

____ source code given
____ tool implemented in a public-domain language
____ not extensible by user
____ other: ____________
____
____
____

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):

____ GUI
____ Library of standard types, functions, and other constructions
____ the library is validated

The extent of the library is (speaking from the point of view of a potential user):
___ not very comprehensive
___ reasonably comprehensive
___ quite comprehensive

____ Editing and document preparation tools
____
____
____

____ Cross-referencing
____ Browsing
____ Requirements tracing
____ Incremental development across multiple sessions
____ Change control and version management
____ Consistency checking
____ Completeness checking
____ Other:
____
____
____
____
____
____
____
____

o How interactive/mechanized/automated is the tool.

____ fully automated
_X__ user guided
____ other: ____________

4. TOOL INPUT AND OUTPUT

o Tool supports these models:

____ synchronous
____ asynchronous
____ mixed

o Input to the tool.

o Output from the tool.

o The language used for input to the tool has (check all that apply):
_X__ formal semantics
_X__ modern programming language constructs (e.g. if-else):
____
____
____
_X__ strong typing
_X__ modularity
_X__ hierarchical design
_X__ parameterization
____ communication between processes
____ buffered
_X__ built-in model of computation
____ other: ____________
____
____

3. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):
____ requirements
_X__ design specification
_X__ implementation
____ test derivation
____ RTL
____ netlists
____ transistor level
____ other: ____________
____

o Has the tool been integrated with other tools?
____ no
_X__ yes - please name tool and applications
with __LSL and LCLint, as mentioned above____________________
with ______________________
with ______________________
____ do not know

4. RESOURCES

o Resource requirements for the tool:
UNIX version _Intel Linux, SPARC SunOS4.1, Solaris 5.3_________
Windows version __________
Mac version ___________

Memory: ____________

o Cost, rights and restrictions:

_X__ free, no license
____ free, license required
  ____ for educational and research use only
____ nominal distribution charge
____ fee for underlying tool(s)
____ flat license fee
____ per user license fee
____ royalties per use
____ other: ___________
____
____

o User background prerequisites (check all that apply):

____ BS degree
_X__ MS degree
____ Ph.D. degree
_X__ knowledge of logic
  _X__ first-order
  ____ high order
____ familiarity with a high-level programming language
____ familiarity with process algebra
____ familiarity with temporal logic
____ other: ___________
____
____

o User's learning curve, if all prerequisites are met:

____ one month
____ two months
_X__ less than six months
____ other
  ______ months

o Tool support

_X__ upgrades/maintenance
  Last version produced at this date: _vs3.1b, 1999____
_X__ manual
  _X__ on the web
____ training
____ listserv
____ mailing list
_X__ dedicated conference(s)/workshop(s)
____ human "help line" _X__ book(s)
_X__ journal/conference publications
_X__ other: __newsgroup comp.specification.larch__________
__ftp archive _____

o Current contact.

http://www.sds.lcs.mit.edu/spd/larch/

6. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

o Verification mechanism(s) (check all that apply):

____ equivalence
____ modal logic
____ temporal logic

____ system or process invariants
____ built-in support for checking for:
  ___ deadlock
  ___ livelock
  ___ other: ____________
  ____
  ____
____ other: ____________
____
____

o Tool supports (check all that apply):
____ optimization and state reduction mechanism using ________________________
____ symbolic simulator:
  ____ interactive
  ____ random
____ feedback on in what state verification failed
  ___ trace leading to the state

7. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

o Degree of proof mechanization.
____ fully mechanized
_X__ partially mechanized

o Support for developing and viewing the proof.

o Presentation of proof to the user (e.g., user input or canonical expressions,
with or without quantifiers).

o Tool supports (check all that apply):
____ automated support for arithmetic reasoning
____ automated support for efficient handling of large propositional expressions
____ automated support for rewriting
____ possible to use lemmas before they are proved.
____ possible to state and use axioms without having to prove them.
____ new definitions can be introduced and existing definitions modified during proof
____ facilities for editing proofs
____ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified
____ reasonably easy to reverify a theorem after slight changes to the specification

8. OPEN-ENDED QUESTIONS

o Strengths of this tool.

o Limitations of this tool.

o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.

o Applications that the tool was used for - case studies, examples, success stories.

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1. http://eis.jpl.nasa.gov/quality/Formal_Methods/

PVS

```
*******************************************************************
****************************     PVS    ***************************
****************************** Sep. 1999*************************
```

For this particular tool, please answer the following questions based on: general description of the tool, tool implementation, tool features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

o Rough classification:
    ____ model checker
    ____ theorem prover
    ____ mechanized proof assistant
    _X__ other:
        Verification system consisting of a specification language and support tools, including a mechanized proof checker integrated with a model checker, ground evaluator, and tabular specification tool.

o Application domain(s) or class(es) of problems originally intended:
    Formalization and verification of requirements and design-level specifications of hardware and software systems.

o Intended audience:
    Anyone interested in formal support for conceptualization and debugging of algorithms, and of software and hardware systems. Both academic and industrial settings.

o Language(s) and/or technique(s) that the tool is based on:
    Classical, typed higher order logic augmented with predicate subtypes, dependent typing, abstract data types, and parameterized theories.

o Reasoning mechanisms used for the tool:
    Low-level decision procedures (including propositional simplification; ground procedures for equality, arithmetic, array, and datatype operations; and model checking) combined with user-definable, high-level proof strategies.
    Sequent Calculus notation.
    CTL model checking using mu-calculus.

o Comparable languages/tools:
    PVS provides more automation than a low-level proof checker (e.g., LCF, HOL, Nuprl, Coq), and more control than a highly automatic theorem prover (e.g., Otter, ACL2). PVS's capabilities are somewhat less generic than Isabelle's.

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation:
    Common Lisp (preferably Franz Inc's Allegro Lisp) with CLOS extensions.
    Emacs or XEmacs (version 19 or later), Tcl/Tk, LaTex.

o How extensible and/or customizable is the tool?
    ____ source code given
    _X__ tool implemented in a public-domain language

____ not extensible by the user
_X__ other:
The PVS environment, including Lisp, Emacs, X windows, and Tcl/Tk, are customizable. Tool makers accept and incorporate suggestions for extending/integrating PVS.

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):
_X__ GUI
_X__ Library of standard types, functions, and other constructions
_X__ the library is validated

The extent of the library is (speaking from the point of view of a potential user):
___ not very comprehensive
_X_ reasonably comprehensive
___ quite comprehensive

_X__ Editing and document preparation tools
_X_ GNU or X Emacs
_X_ Customized prettyprinting and typesetting (using LaTex)
____ Cross-referencing
_X__ Browsing
____ Requirements tracing
_X__ Incremental development across multiple sessions
____ Change control and version management
_X__ Consistency checking
_X__ Completeness checking
____ Other

o How interactive/mechanized/automated is the tool?
____ fully automated
_X__ user guided
(simpler steps are automated)
_X__ other:
__The user may also define application-specific strategies to automate the verification.

4. TOOL INPUT AND OUTPUT

o Tool supports these models:
____ synchronous
____ asynchronous
_X__ mixed:

o Input to the tool:
ASCII text consisting of a specification in the PVS language.

o Output from the tool:
Proof results, status information, alltt and latex output, specification files, proof files.

o The language used for input to the tool has (check all that apply):
_X__ formal semantics
_X__ modern programming language constructs(e.g. if-else):
__if-else, let, where

__structured datatypes (e.g., records, tuples, ennumerations)
__abstract data types
__tabular notation
_X__ strong typing
_X__ modularity
____ hierarchical design
_X__ parameterization
____ communication between processes
____ buffered
____ built-in model of computation
_X__ other:
__Undecidable typechecking: to cope with this, the typechecker generates proof obligations, most of which are discharged automatically by the prover.
__Overloading: PVS allows a liberal amount of overloading.
__Automated support for judgements and coercions (conversions).
__Total vs partial functions: in PVS, functions represent total maps; partial functions are admitted within this framework via the predicate subtype mechanism.

5. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):
_X__ requirements
_X__ design specification
_X__ implementation
_X__ test derivation
_X__ RTL
____ netlists
____ transistor level
_X__ other:
__mathematics
(in principle, every level can be addressed, but some levels require more work than others)

o Has the tool been integrated with other tools?
____ no
_X__ yes:
__model-checker (Janssen's BDD-based model checker for the propositional mu-calculus __Technical Univ. of Eindhoven)
__TAME (Lynch-Vaandrager Timed Automata system models __NRL)
__SCR* (Software Cost Reduction method __NRL)
__InVest (Tool for automatic invariant generation __Verimag)
__Pamela (VDM-style verification system __Univ. of Bremen)
__Mona (language/tool for monadic second order logic __BRICS)
__SVC (Stanford Validity Checker for subset of first-order logic __Stanford University)

6. RESOURCES

o Resource requirements for the tool:
UNIX version: __precompiled for Solaris 2 or higher (SPARC workstations),
Redhat Linux
Windows version __________

Mac version __________
Memory: 20 mb disk space, 50 mb swap space, 32 mb real memory

o Cost, rights and restrictions:
 ____ free, no license
 _X__ free, license required
 ____ for educational and research use only
 ____ nominal distribution charge
 ____ fee for underlying tool(s)
 ____ flat license fee
 ____ per user license fee
 ____ royalties per use
 ____ other

o User background prerequisites (check all that apply):
 _X__ BS degree
 ____ MS degree
 ____ Ph.D. degree
 _X__ knowledge of logic
 _X__ first-order
 ____ high order
 _X__ familiarity with a high-level programming language
 ____ familiarity with process algebra
 ____ familiarity with temporal logic
 ____ other:

o User's learning curve, if all prerequisites are met:
 ____ one month
 ____ two months
 ____ less than six months
 _X__ other
 __6__ months

o Tool support:
 _X__ upgrades/maintenance
 Last version produced at this date: PVS 2.3, 1999
 _X__ manual
 _X__ on the web
 _X__ training
 (tutorials on the web)
 ____ listserv
 _X__ mailing list
 ____ dedicated conference(s)/workshop(s)
 ____ human "help line"
 ____ book(s)
 _X__ journal/conference publications
 _X__ other:
 __bugs, problems, suggestions to pvs-bugs@csl.sri.com
 __list of user suggestions and SRI's responses on the web
 __archive, FAQ, libraries on the web

o Contact:
 pvs-request@csl.sri.com
 http://pvs.csl.sri.com

7. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

o Verification mechanism(s) (check all that apply):
_X__ equivalence
_X__ modal logic
_X__ temporal logic
(CTL and fair CTL)
_X__ system or process invariants
_X__ built-in support for checking for:
_X_ deadlock
_X_ livelock
_X_ boolean propositions
_X_ other: __fairness__
____ other

o Tool supports (check all that apply):
_X__ optimization and state reduction mechanism
____ simulator
____ interactive
____ random
_x__ feedback on state in which verification failed
(Counterexample generation is currently under development.)

8. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

o Degree of proof mechanization:
____ fully mechanized
_X__ partially mechanized
(although finite state verification and the proof of many straightforward results are fully automatic. There is also a batch mode in which proofs may be easily rerun, and a facility for defining proof strategies to automate proofs.

o Support for developing and viewing the proof:
Tcl/Tk interface to display proof trees and theory hierarchies.
Proofs yield scripts that may be edited, attached to additional formulas,
and rerun. Proofs may also be checkpointed, providing rapid access to parts of a proof the user wishes to examine or adjust.

o Presentation of proof to the user (e.g., user input or canonical expressions
with or without quantifiers):
Proofs are presented in a sequent-style representation. PVS takes care to assure that the initial proof goal transparently reproduces the formula input by the user. Quantification is retained; implicit universal quantification in the user's specification is made explicit.

o Tool supports (check all that apply):
_X__ automated support for arithmetic reasoning
_X__ automated support for efficient handling of large propositional expressions

_X__ automated support for rewriting
_X__ possible to use lemmas before they are proved.
_X__ possible to state and use axioms without having to prove them.
_X__ new definitions can be introduced and existing definitions modified during proof
_X__ facilities for editing proofs
_X__ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified
_X__ reasonably easy to reverify a theorem after slight changes to the specification
_X__ other:
  __integration with CTL model checking
  __ground evaluator (providing "run" speeds comparable to imperative programs)
  __proof strategies
  __proof storage, replay, and checkpointing
  __graphical display of proof trees, theory hierarchies, and prover commands
  __proof chain analysis
  __proof and theory status reporting

9. OPEN-ENDED QUESTIONS

o Strengths of this tool:
Comprehensive, interactive environment for writing formal specifications and checking formal proofs, including tight integration of
algorithmic and deductive proof technologies.
Generic system well suited to, e.g., prototyping specialized strategies, embedding logics, and exploring strategies for integrating formal techniques, as well as to undertaking proofs of difficult algorithms and complex systems.

o Limitations of this tool:
PVS's capabilities complement, but do not compete with those of dedicated
lightweight tools for specialized applications.
Not industrial strength, but a mature research prototype.
User learning curve.

o Estimated possible uses of the tool (e.g., applications, classes of problems, stages of production cycle):
Hardware verification, embedding logics, fault-tolerant algorithms, library development, invariant generation and abstraction, distributed algorithms, requirements specification and verification, security protocols, test generation.

o Applications that the tool was used for - case studies, examples, success stories:

Posted on http://pvs.csl.sri.com. Examples:
Hardware:
  __Collins Commercial Avionics microprocessor design
  __Fujitsu high level design and validation of ATM switch

__NASA single pulser digital circuit
__IEEE 854 floating point standard
__SRT division

Distributed Algorithms:

- __FLASH cache coherence protocol
- __bounded retransmission protocol
- __real-time controllers

Fault Tolerant Algorithms:

- __Fault-tolerant agreement and diagnosis protocols for various architectures and fault models

Embedding Logics:

- __Duration calculus
- __The B-method
- __A real-time Hoare logic

Invariant Generation and Abstraction:

- __PVS has been used as a simplifier in several systems for the heuristic discovery of loop invariants for distributed protocols

Requirements:

- __Space Shuttle flight software
- __Cassini spacecraft fault-protection software

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1. http://eis.jpl.nasa.gov/quality/Formal_Methods/

Z/EVES

```
*************************************************************************
************************************Z/EVES*****************************
************************************Sep. 1999*************************
```

For this particular tool, please answer the following questions grouped based on: general description of the tool, tool implementation, tool features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

o Rough classification:
____ model checker
____ theorem prover
_X__ mechanized proof assistant
____ other: __Z interface to EVES mechanized proof assistant.___

o Application domain(s) or class(es) of problems originally intended.
Analytical support for writers of Z specifications.
Formal methods courses.
Various applications in safety- and security- domains.

o Intended audience.
Students, lecturers, researchers, commercial users interested in rigorous
specifications supported by rigorous analysis.

o Language(s) and/or technique(s) that the tool is based on.
Z, Verdi, s-Verdi.
Verdi is a language based on untyped set theory.

o Reasoning mechanisms used for the tool.
General theorem proving, specifying and implementing programs, proving consistency between specification and implementation.
Syntax and type checking, schema expansion, domain checking, pre-condition calculation, refinement, and general conjectures about a specification.
EVES has a programming component and supports pre/post proofs, in addition to general mathematical modeling.

o Comparable languages/tools.
ProofPower, Cadiz and Zola.

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation.
Implemented in Lisp.

o How extensible and/or customizable is the tool.
____ source code given
_X__ tool implemented in a public-domain language
_X__ not extensible by user
____ other: ____________

APIs are now defined for Z/EVES allowing for interchanges between tools.

Plans are to augment Z/EVES with 3rd party developments. Currently, only
executables are distributed.

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):

_X__ GUI

_X__ Library of standard types, functions, and other constructions

__X_ the library is validated

The extent of the library is (speaking from the point of view of a potential user):

___ not very comprehensive

__ reasonably comprehensive

_X_ quite comprehensive

It contains all of the Spivey toolkit, which is the general basis for all Z specifications.

_X__ Editing and document preparation tools

_Framemaker-based Z editor____________

_Framemaker editor that has an API connection to Z/EVES.

____

____

____ Cross-referencing

__X_ Browsing

(to be completed soon)

____ Requirements tracing

__X_ Incremental development across multiple sessions

(to be completed soon)

____ Change control and version management

__X_ Consistency checking

__X_ Completeness checking

_X__ Other:

__syntax and type checking____________

__schema expansion____________

__precondition calculation____________

__domain checking____________

__proving consistncy between specification and implementation___

__support for the Mathematical Toolkit as described in Spivey's 2nd edition of "The Z Notation"____

____

____

o How interactive/mechanized/automated is the tool.

____ fully automated

_X__ user guided

some prover steps are automated

____ other: ___________

4. TOOL INPUT AND OUTPUT

o Tool supports these models:

____ synchronous

____ asynchronous

____ mixed

o Input to the tool.

Z, Verdi or s-Verdi specification.

o Output from the tool.

Proof results.

o The language used for input to the tool has (check all that apply):

Note: the following paragraph refers to Verdi language:

_X__ formal semantics
_X__ modern programming language constructs (e.g. if-else):
____
____
____
_X__ strong typing
_X__ modularity
_X__ hierarchical design
_X__ parameterization
_X___ communication between processes
____ buffered
____ built-in model of computation
____ other: ____________
____

3. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):

_X__ requirements
_X__ design specification
_X__ implementation
____ test derivation
____ RTL
____ netlists
____ transistor level
_X__ other: __mathematics__________
____

o Has the tool been integrated with other tools?

____ no
_X__ yes - please name tool and applications
with __Z browser, supplies text input to Z/EVES___
with __Z-browser plug-in, for displaying Z notation using Netscape; runs on Windows 95/NT__
with __Z Abstract Syntax Tree Viewer, to display abstract syntax trees of Z specifications; runs on Windows 95/NT__
with __Zeus (Framemaker editor)__
with __RoZ (an environment integrating UML and Z)__
with __Z animator (work in progress)__
____ do not know

4. RESOURCES

o Resource requirements for the tool:

UNIX version __SunOS 4.x, Linux ELF________
Windows version __3.1,95,98,NT________
Mac version ____________
Memory: __at least 32Mb___________

o Cost, rights and restrictions:

____ free, no license
_X__ free, license required
___ for educational and research use only
____ nominal distribution charge
____ fee for underlying tool(s)
____ flat license fee
____ per user license fee
____ royalties per use
____ other: ____________

o User background prerequisites (check all that apply):
_X__ BS degree
____ MS degree
____ Ph.D. degree
_X__ knowledge of logic
_X__ first-order
____ high order
____ familiarity with a high-level programming language
____ familiarity with process algebra
____ familiarity with temporal logic
_X__ other: __The above checked fields refer to performing proofs. Type checking, schema expansion, pre-condition calculation, domain checking without proof, require no knowledge of logic.__________

o User's learning curve, if all prerequisites are met:
____ one month
____ two months
____ less than six months
____ more than six months
______ months

Note: Depends upon application. Type checking, schema expansion, pre-condition calculation, and domain checking (without proof) should only take a day or two to learn. Learning to preform more serious proofs could take several months.

o Tool support
_X__ upgrades/maintenance
Last version produced at this date: _vs.3x, due November 1999_____
_X__ manual
_X__ on the web
_X__ training
Course is provided.
_X__ listserv
____ mailing list
_X__ conference(s)/workshop(s)
_X__ human
ORA will provide consulting.
____ book(s)
_X__ journal/conference publications
____ other: ____________

o Current contact.
http://www.ora.on.ca/z-eves/

zeves@ora.on.ca (subscribe at zeves-request@ora.on.ca)

6. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

o Verification mechanism(s) (check all that apply):
  ____ equivalence
  ____ modal logic
  ____ temporal logic
  ____ system or process invariants
  ____ built-in support for checking for:
    ___ deadlock
    ___ livelock
    ___ other: ___________
  ____ other: ___________
  ____

o Tool supports (check all that apply):
  ____ optimization and state reduction mechanism using ______________________
  ____ symbolic simulator:
    ____ interactive
    ____ random
  ____ feedback on in what state verification failed
    ___ trace leading to the state

7. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

o Degree of proof mechanization.
  ____ fully mechanized
  _X__ partially mechanized

o Support for developing and viewing the proof.
  Proof browsing.

o Presentation of proof to the user (e.g., user input or canonical expressions,
with or without quantifiers).
Z-like notation.

o Tool supports (check all that apply):
  _X__ automated support for arithmetic reasoning
  _X__ automated support for efficient handling of large propositional expressions
  _X__ automated support for rewriting
  _X__ possible to use lemmas before they are proved.
  _X__ possible to state and use axioms without having to prove them.
  _X__ new definitions can be introduced and existing definitions modified during proof

Would have to restart the proof.
  _X__ facilities for editing proofs
  _X__ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified
  _X__ reasonably easy to reverify a theorem after slight changes to the specification

8. OPEN-ENDED QUESTIONS

o Strengths of this tool.

Rigorously developed SPARC Verdi compiler for EVES/Verdi.
Synergy of an expressive writable notation (Z) with an automated Analytical engine. Useful for the Z community.

o Limitations of this tool.

Limited to Z community, can take long time to learn.

o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.

Education, safety, security.

o Applications that the tool was used for - case studies, examples, success stories.

Some are posted on http://www.ora.on.ca/biblio-welcome.html.
Analysis of authentication protocols, including X.509.
Design of a prototype High Assurance One-Way Link.
Many proprietary applications.

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1. http://eis.jpl.nasa.gov/quality/Formal_Methods/

Concurrency Factory

```
*******************************************************************
********************** CONCURRENCY FACTORY ***********************
****************************Sep. 1999******************************

For this particular tool, please answer the following questions grouped
based on: general description of the tool, tool implementation, tool
features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

o Rough classification:
      ____ model checker
      ____ theorem prover
      ____ mechanized proof assistant
      _X__ other: _integrated toolset: model checker, simulators,
                   graphical and textual user interface, code
                   generator_______
o Application domain(s) or class(es) of problems originally intended.
      Concurrent systems, such as protocols or control systems;
      networks of finite-state processes.
      Industrial problems, e.g. in telecommunications industry.
o Intended audience.
      Protocol engineers and software developers.
o Language(s) and/or technique(s) that the tool is based on.
      GCCS, a graphical variant of the process algebra CCS aimed at
      specifying hierarchical networks of processes.
      VPL, a textual language for hierarchical networks of processes,
      with support for complex data and control structures.
o Reasoning mechanisms used for the tool.
      Computing set of transitions possible for a system in a given state
      using formal operational semantics.
      GCCS interpreted by all the tools in the toolkit.
o Comparable languages/tools.
      CWB, Spin.

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation.
      C++, Tcl/Tk.
o How extensible and/or customizable is the tool.
      ____ source code given
      _X__ tool implemented in a public-domain language
      ____ not extensible by user
      ____ other: ____________

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):
      _X__ GUI
            __for GCCS__
```

____ Library of standard types, functions, and other constructions
____ the library is validated

The extent of the library is (speaking from the point of view of a potential user):
___ not very comprehensive
___ reasonably comprehensive
___ quite comprehensive

_X__ Editing and document preparation tools
_textual user interface for VPL__________

____

____

____ Cross-referencing
____ Browsing
____ Requirements tracing
____ Incremental development across multiple sessions
____ Change control and version management
____ Consistency checking
____ Completeness checking
____ Other:
__graphical compiler for generating Facile code (similar to Standard ML and CCS), Java and Ada'95 code.____________
__graphical simulators for GCCS
__simulator for VPL____________

____

____

o How interactive/mechanized/automated is the tool.
_X__ fully automated
____ user guided
____ other: ____________

4. TOOL INPUT AND OUTPUT

o Tool supports these models:
____ synchronous
_X__ asynchronous
____mixed

o Input to the tool.
GCCS or VPL specification or combination of the two.

o Output from the tool.
Step 1: networks of finite-state processes.
Step 2: model checking and/or code generation.

o The language used for input to the tool has (check all that apply):
GCCS:
_X__ formal semantics
____ modern programming language constructs (e.g. if-else):

____

____

____

____ strong typing
____ modularity
_X__ hierarchical design
____ parameterization

_X__ communication between processes
____ buffered
____ built-in model of computation
_X__ other: _graphical___________
_based on CCS___
____

VPL:
_X__ formal semantics
_X__ modern programming language constructs (e.g. if-else):
__integers of limited size
__arrays and records of integers__________
__if-then-else__
__while-do__
__select___
____
____
____ strong typing
____ modularity
____ hierarchical design
_X__ parameterization
____ communication between processes
____ buffered
____ built-in model of computation
_X__ other: __finite data domain__________
____
____

3. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):
____ requirements
_X__ design specification
_X__ implementation
(code generation)
____ test derivation
____ RTL
____ netlists
____ transistor level
____ other: _____________
____

o Has the tool been integrated with other tools?
_X__ no
____ yes
with ______________________
____ do not know

4. RESOURCES

o Resource requirements for the tool:
UNIX version __SunOS 4.1 or Solaris on Sun SPARC__
Windows version __________
Mac version ___________
Memory: _____________

o Cost, rights and restrictions:
____ free, no license
_X__ free, license required
    _X__ for educational and research use only
____ nominal distribution charge
____ fee for underlying tool(s)
____ flat license fee
____ per user license fee
____ royalties per use
____ other: ____________

o User background prerequisites (check all that apply):
_X__ BS degree
____ MS degree
____ Ph.D. degree
____ knowledge of logic
    ____ first-order
    ____ high order
_X__ familiarity with a high-level programming language
____ familiarity with process algebra
____ familiarity with temporal logic
____ other: ____________
____

o User's learning curve, if all prerequisites are met:
_X__ one month
____ two months
____ less than six months
____ other
    ______ months

o Tool support
_X__ upgrades/maintenance
    Last version produced at this date: _1998_
    New version to be released in near future.
____ manual
    ____ on the web
____ training
____ listserv
____ mailing list
____ dedicated conference(s)/workshop(s)
____ human "help line"
____ book(s)
_X__ journal/conference publications
____ other: ____________
____

o Current contact.
concurr@cs.sunysb.edu
http://www.cs.sunysb.edu/~concurr

6. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

o Verification mechanism(s) (check all that apply):
____ equivalence
_X__ modal logic
    _linear-time local and global model checker for alteration-free

modal mu-calculus
_local model checker for real-time extension for the above logic
____ temporal logic
____ system or process invariants
____ built-in support for checking for:
___ deadlock
___ livelock
___ other: ____________
____

_X__ other: __strong and weak bisimulation____________
____
____

o Tool supports (check all that apply):
LMC (local model checker):
_X__ optimization and state reduction mechanism
using __on-the-fly execution and partial order reduction_
____ simulator:
____ interactive
____ random
__X_ feedback on in what state verification failed
_X_ trace leading to the state
(if the user chooses so)

7. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

o Degree of proof mechanization.
____ fully mechanized
____ partially mechanized
o Support for developing and viewing the proof.
o Presentation of proof to the user (e.g., user input or canonical expressions,
with or without quantifiers).
o Tool supports (check all that apply):
____ automated support for arithmetic reasoning
____ automated support for efficient handling of large propositional expressions
____ automated support for rewriting
____ possible to use lemmas before they are proved.
____ possible to state and use axioms without having to prove them.
____ new definitions can be introduced and existing definitions modified during proof
____ facilities for editing proofs
____ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified
____ reasonably easy to reverify a theorem after slight changes to the specification

8. OPEN-ENDED QUESTIONS

o Strengths of this tool.
Designed for use by protocol engineers and software developers, for industrial-scale problems.
Specification, simulation, verification and code generation of concurrent systems modeled as hierarchical networks of finite-state processes.

Sophisticated graphical support for specification and simulation.
Automatic code generation from verified specifications.

o Limitations of this tool.
Finite-state systems.

o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.
Main application area is reactive systems, including embedded system software, process control systems, telecommunication protocols, security protocols, and e-commerce protocols.

o Applications that the tool was used for - case studies, examples, success stories.
Posted on http://www.cs.sunysb.edu/~concurr/. Examples:
Specification and verification of: GNU UUCP i-Protocol, E-2C Hawkeye Early Warning Aircraft Display LAN Protocol, RETHER real-time Ethernet protocol.

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1. http://eis.jpl.nasa.gov/quality/Formal_Methods/

Murphi

```
***********************************************************************
**************************** MURPHI **********************************
**************************** Sep. 1999********************************

For this particular tool, please answer the following questions grouped
based on: general description of the tool, tool implementation, tool
features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

o Rough classification:
      _X__ model checker
      ____ theorem prover
      ____ mechanized proof assistant
      ____ other: ________________
o Application domain(s) or class(es) of problems originally intended.
      Hardware protocol verification, optional extensions for cryptographic
      protocols.
      Early design stages, error finding.
o Intended audience.
      Digital designers.
o Language(s) and/or technique(s) that the tool is based on.
      Murphi language: collection of guarded rules (condition/action),
      which are executed repeatedly in an infinite loop (similar to
      Chandy and Misra's Unity language.)
o Reasoning mechanisms used for the tool.
      Explicit state space enumeration, depth- or breath- first search;
      simulation.
o Comparable tools:
      SMV, Spin, Concurrency Factory, CWB.

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation.
      C++.
o How extensible and/or customizable is the tool.
      _X__ source code given
      _X__ tool implemented in a public-domain language
      ____ not extensible by user
      ____ other: ____________
                  ____________

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):
      ____ GUI
      ____ Library of standard types, functions, and other constructions
            ____ the library is validated

            The extent of the library is (speaking from the point of view of
```

a potential user):
___ not very comprehensive
___ reasonably comprehensive
___ quite comprehensive

Note: while there is no standard library, a number of types and functions that are commonly provided by a library are provided in the language, for example, arrays, records, Multiset and Scalarset.

____ Editing and document preparation tools
____
____
____
____ Cross-referencing
____ Browsing
____ Requirements tracing
____ Incremental development across multiple sessions
____ Change control and version management
____ Consistency checking
____ Completeness checking
____ Other:
____
____
____
____
____

o How interactive/mechanized/automated is the tool.
_X__ fully automated
____ user guided
____ other: ____________

4. TOOL INPUT AND OUTPUT

o Tool supports these models:
____ synchronous
_X__ asynchronous
(interleaving)
____ mixed

o Input to the tool.
Murphi description.

o Output from the tool.
If a boolean invariant is violated, error message and error trace.
Reports if error or assertion statements are reached.

o The language used for input to the tool has (check all that apply):
_X__ formal semantics
_X__ modern programming language constructs (e.g. if-else):
__if_________
__switch__________
__for__________
__while_______
____ strong typing
____ modularity
____ hierarchical design
_X__ parameterization

____ communication between processes
    ____ buffered
____ built-in model of computation
____ other: ____________
____
____

5. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):
____ requirements
_X__ design specification
____ implementation
____ test derivation
____ RTL
____ netlists
____ transistor level
____ other: ____________
____

o Has the tool been integrated with other tools?
____ no
_X__ yes
with ___SVC__________________
with _____________________
____ do not know

6. RESOURCES

o Resource requirements for the tool:
UNIX version __precompiled for: INDY IRIX 5.3,
SunSPARC20 SunOS 4.1.3_U1, 4.1.4, 5.4,
SunSPARCserver-1000 SunOS 5.5,
Intel Linux 1.3.48, 2.0.27, 2.0.34, 2.0.36_
Windows version __________
Mac version ___________
Memory: _____________

o Cost, rights and restrictions:
____ free, no license
_X__ free, license required
(however, user does not have to send in anything)
    ____ for educational and research use only
____ nominal distribution charge
____ fee for underlying tool(s)
____ flat license fee
____ per user license fee
____ royalties per use
____ other: ____________

o User background prerequisites (check all that apply):
_X__ BS degree
____ MS degree
____ Ph.D. degree
____ knowledge of logic

____ first-order
____ high order
_X__ familiarity with a high-level programming language
____ familiarity with process algebra
____ familiarity with temporal logic
____ other: ____________

o User's learning curve, if all prerequisites are met:
_X__ one month
____ two months
____ less than six months
____ other:
______ months

o Tool support
_X__ upgrades/maintenance
Last version produced at this date: _Murphi 3.1, 1999
_X__ manual
_X__ on the web
____ training
____ listserv
_X__ mailing list
____ dedicated conference(s)/workshop(s)
____ human "help line"
____ book(s)
_X__ journal/conference publications
_X__ other: __bug reports, suggestions to murphi@verify.stanford.edu

o Current contact.
http://sprout.stanford.edu/dill/murphi.html
murphi@verify.stanford.edu

7. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

o Verification mechanism(s) (check all that apply):
____ equivalence
____ modal logic
____ temporal logic
_X__ system or process invariants (boolean propositions true for all states of the system/process)
____ built-in support for checking for:
_X_ deadlock
___ livelock
___ other: __error statements__________
__assertion statements_________

____ other: ____________

o Tool supports (check all that apply):
_X__ optimization and state reduction mechanism
state reduction using:
__symmetry (description has identical elements that

can be permuted consistently without changing
verification properties)
__reversible rules (condition/action can be executed "in
reverse")
__repetition constructors (keeping track of how many
processes
are in the same state)
__hash compression algorithms for probabilistic
verification
optimization using:
__probabilistic verification
__state space caching
__parallel Murphi
__using magnetic disk instead of main memory
____ simulator:
____ interactive
__X_ random
_X__ feedback on in what state verification failed
_X_ trace leading to the state
____ other:
____

8. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

o Degree of proof mechanization.
____ fully mechanized
____ partially mechanized
o Support for developing and viewing the proof.
o Presentation of proof to the user (e.g., user input or canonical
expressions,
with or without quantifiers).
o Tool supports (check all that apply):
____ automated support for arithmetic reasoning
____ automated support for efficient handling of large propositional
expressions
____ automated support for rewriting
____ possible to use lemmas before they are proved.
____ possible to state and use axioms without having to prove them.
____ new definitions can be introduced and existing definitions
modified during proof
____ facilities for editing proofs
____ the foundations (i.e., all axioms, definitions, assumptions,
lemmas) of the proof are identified
____ reasonably easy to reverify a theorem after slight changes to
the specification

9. OPEN-ENDED QUESTIONS

o Strengths of this tool.
Designed for industrial use by non-experts in formal methods.
Optimization and state reduction algorithms and techniques.
o Limitations of this tool.
No checking for liveness and fairness properties (e.g. livelock).

No message communication.
Not possible to describe sequential behavior.

o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.

Multiprocessor cache coherence protocols. Security protocols.

o Applications that the tool was used for - case studies, examples, success stories.

Listed at http://sprout.stanford.edu/dill/murphi.html. Examples:
Verification of cache coherence protocols for Sun UltraSparc-1
Verification of cache coherence and link level protocol for Sun's S3.mp multiprocessor
Specification and verification of Sparc V9 TSO, PSO, RMO memory models
Cryptographic and security protocols
Verification of a part of "Scalable Coherent Interface" IEEE Std 1596-1992
Proprietary industrial protocols, for Fujitsu, HAL Computer Systems, HP, IBM, ad others

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1. http://eis.jpl.nasa.gov/quality/Formal_Methods/

SVM Cadence

```
**********************************************************************
******************** SMV Cadence Berkeley Labs *********************
**************************** Sep. 1999******************************

For this particular tool, please answer the following questions grouped
based on: general description of the tool, tool implementation, tool
features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

o Rough classification:
        __X_ model checker
        ____ theorem prover
        __X_ mechanized proof assistant
           (of limited scope and built on top of the model checker)
        ____ other: _________________
o Application domain(s) or class(es) of problems originally intended.
      Hardware verification.
o Intended audience.
      General.
o Language(s) and/or technique(s) that the tool is based on.
      SMV input language is used to describe a refinement hierarchy (that
      is, specifications at multiple levels of abstraction).
      Specifications are written in temporal logic, or an HDl-like
      equational notation. It is also possible to input models in a
      synchronous version of the Verilog HDL. The logic is effectively
      a first-order, quantifier free, linear time temporal logic.
o Reasoning mechanisms used for the tool.
      Model checking (determines the truth of temporal formulas by exhaustive
      state-space exploration).
o Comparable languages/tools.
      Spin, the Concurrency Workbench, the Concurrency Factory, VIS,
      Mocha, COSPAN, FormalCheck.
      This tool is an extension of Carnegie Mellon SMV to support
      compositional methods.
      Note: SMV is a research vehicle, and is not directly related the
      FormalCheck product from Cadence.

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation.
      OBDD-based model checking algorithm, implemented in C language.
      Compositional proof methods, also implemented in C.
o How extensible and/or customizable is the tool.
        ____ source code given
        ____ tool implemented in a public-domain language
        _X__ not extensible by the user.
        ____ other:    ____________

3. TOOL FEATURES AND UTILITIES
```

o Tool supports the following (check all that apply):
_X__ GUI
____ Library of standard types, functions, and other constructions
____ the library is validated
The extent of the library is (speaking from the point of view of
a potential user):
___ not very comprehensive
___ reasonably comprehensive
___ quite comprehensive
Note: while there is no standard library, a number of types and functions that are commonly provided by a library are provided in the language, for example, bit vectors and binary arithmetic, arrays, structures. Queues are notably absent, however.
_X__ Editing and document preparation tools
__Emacs interface____________
____
____ Cross-referencing
_X__ Browsing
____ Requirements tracing
____ Incremental development across multiple sessions
____ Change control and version management
____ Consistency checking
____ Completeness checking
_X__ Other:
__BDD library (implemented in C) for sequential verification_
__support for refinement verification_____________
____
____
____

o How interactive/mechanized/automated is the tool.
__X_ fully automated
__X_ user guided
(User guidance is required for refinement verification.)
____ other: ____________

4. TOOL INPUT AND OUTPUT

o Tool supports these models:
_X__ synchronous
_X__ asynchronous
_X__ mixed

o Input to the tool.
Model in SMV language (a collection of properties expressed in temporal logic) or Synchronous Verilog (which is then translated into SMV language).

o Output from the tool.
Yes/no answer to posed temporal formulas, counterexample if "no." Also, keeps track of the status of proof obligations in compositional proofs.

o The language used for input to the tool has (check all that apply):
____ formal semantics

_X__ modern programming language constructs (e.g. if-else):
control constructs: if/else, while, forall, default
data types: scalars, enumerated types, structures, arrays
_X__ strong typing
(typing is used only to enforce symmetry)
_X__ modularity
_X__ hierarchical design
_X__ parameterization
(can describe designs with arbitrary number of components, etc.)
_X__ communication between processes
(signals and shared variables)
____ buffered
____ built-in model of computation
____ other: ____________
____

5. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):
____ requirements
_X__ design specification
_X__ implementation
____ test derivation
_X__ RTL
_X__ netlists
____ transistor level
____ other: ____________
____

o Has the tool been integrated with other tools?
____ no
_X__ yes - please name tool and applications
with _bounded model checker form CMU_
with ____________________
with ____________________
____ do not know

6. RESOURCES

o Resource requirements for the tool:
UNIX version ___Intel 386 Linux, SPARC SunOS, Solaris, HPUX, MIPS/Irix.
Windows version __NT, 95________
Mac version ___________
Memory: _____________
o Cost, rights and restrictions:
____ free, no license
__X_ free, license required
_X__ for educational and research use only
____ nominal distribution charge
____ fee for underlying tool(s)
____ flat license fee
____ per user license fee
____ royalties per use

____ other: ____________

o User background prerequisites (check all that apply):

_X__ BS degree
____ MS degree
____ Ph.D. degree
____ knowledge of logic
    ____ first-order
    ____ high order
____ familiarity with a high-level programming language
____ familiarity with process algebra
_X__ familiarity with temporal logic
____ other: ____________
____
____

o User's learning curve, if all prerequisites are met:

____ one month
_X__ two months
____ less than six months
____ other
    ______ months

o Tool support

_X__ upgrades/maintenance
    Last version produced at this date: __1999___
_X__ manual
    _X__ on the web
_X__ training
    lecture notes and tutorials, on the web
____ listserv
_X__ mailing list
____ dedicated conference(s)/workshop(s)
____ human "help line"
____ book(s)
____ journal/conference publications
_X__ other: _archive and FAQ, on the web_
    _questions and comments to smv-users@cadence.com

o Current contact.

http://www.cs.cmu.edu/~modelcheck/index.html for older version of SMV

http://www.cis.ksu.edu/santos/smv-doc/
http://www-cad.eecs.berkeley.edu/~kenmcmil/
smv-users@cadence.com

7. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

o Verification mechanism(s) (check all that apply):

____ equivalence
____ modal logic
_X__ temporal logic
    __CTL, LTL__
____ system or process invariants
____ built-in support for checking for:
    ___ deadlock

___ livelock
___ other: ____________

____
____ other: ____________
____

o Tool supports (check all that apply):
_X__ optimization and state reduction mechanism
using _ compositional methods: data type reduction, uninterpreted functions, cone-of-influence reduction, temporal case splitting, constant propagation, circular compositional proofs, symmetry reductions, induction over the natural numbers, refinement verification.
_X__ simulator:
____ interactive
_____ random
_X__ feedback on in what state verification failed
_X_ trace leading to the state

8. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

Note: SMV is not a general purpose theorem prover, but it does provide a special-purpose proof assistant.

o Degree of proof mechanization.
____ fully mechanized
_X__ partially mechanized
o Support for developing and viewing the proof.
Graphical browser.
o Presentation of proof to the user (e.g., user input or canonical expressions,
with or without quantifiers).
None.
o Tool supports (check all that apply):
_X__ automated support for arithmetic reasoning (limited to modular, binary arithmetic)
_X__ automated support for efficient handling of large propositional expressions
____ automated support for rewriting
_X__ possible to use lemmas before they are proved.
_X__ possible to state and use axioms without having to prove them.
_X__ new definitions can be introduced and existing definitions modified during proof
_X__ facilities for editing proofs
_X__ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified
_X__ reasonably easy to reverify a theorem after slight changes to the specification

9. OPEN-ENDED QUESTIONS

o Strengths of this tool.

Combines model checking and compositional proof methods.
This means that, on the one hand, the state explosion
problem can be avoided by decomposition, while on the other hand, model checking can be used to avoid writing detailed invariants by hand.

o Limitations of this tool.

Not user-extensible, in the way that most proof assistants are.
Limited to first-order temporal logic.

o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.

Hardware verification.

o Applications that the tool was used for - case studies, examples, success stories.

Verification of the RTL-level implementation of a cache coherence protocol
(SGI), as well as numerous cache coherence protocols at an abstract level.
Verification of standard hardware protocols, e.g. Futurebus+ and PCI local bus protocols.
Numerous applications in low-level hardware verification.

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1.
http://eis.jpl.nasa.gov/quality/Formal_Methods/

SPIN

```
***********************************************************************
****************************   Spin   ********************************
******************************Sep. 1999*******************************
```

For this particular tool, please answer the following questions grouped based on: general description of the tool, tool implementation, tool features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

o Rough classification:
    _X__ model checker
    ____ theorem prover
    ____ mechanized proof assistant
    ____ other: ______________

o Application domain(s) or class(es) of problems originally intended.
    Software, distributed systems.

o Language(s) and/or technique(s) that the tool is based on.
    PROMELA (PROcess MEta LAnguage), a non-deterministic language loosely based on Dijkstra's guarded command language notation, and borrowing the notation for I/O operations from Hoare's CSP language.

o Reasoning mechanisms used for the tool.
    State space exploration (exhaustive or partial); simulation.

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation.
    ANSI C, on-the-fly checking.

o How extensible and/or customizable is the tool.
    _X__ source code given
    _X__ tool implemented in a public-domain language
    ____ not extensible by user
    ____ other: __________
                ____
                ____

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):
    _X__ GUI
        (Xspin)
    ____ Library of standard types, functions, and other constructions
        ___ the library is validated

        The extent of the library is (speaking from the point of view of a potential user):
        ___ not very comprehensive
        ___ reasonably comprehensive
        ___ quite comprehensive

Note: while there is no standard library, a number of types and functions that are commonly provided by a library are provided in the language, for example, arrays and queues.

____ Editing and document preparation tools

____
____
____

____ Cross-referencing
____ Browsing
____ Requirements tracing
____ Incremental development across multiple sessions
____ Change control and version management
_X__ Consistency checking
_X__ Completeness checking
____ Other:
__depository of source code extensions on SPIN web page______________

____

o How interactive/mechanized/automated is the tool.

_X__ fully automated
_X__ user guided
(simulation option)
____ other: ____________

4. TOOL INPUT AND OUTPUT

o Tool supports these models:

____ synchronous
_X__ asynchronous
(interleaving)
____ mixed

o Input to the tool.

Model written in PROMELA (somewhat resembles a C program).

o Output from the tool.

Yes/no answer to posed tests;
trace leading to errors;
% coverage of state space.

o The language used for input to the tool has (check all that apply):

_X__ formal semantics
_X__ modern programming language constructs (e.g. if-else):
__if-else__________
__do___________
____
____ strong typing
____ modularity
____ hierarchical design
_X__ parameterization
_X__ communication between processes
__X__ buffered
__X__ rendezvous
__X__ through shared memory
_X__ built-in model of computation

____ other: ____________
____
____

5. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):
_X__ requirements
_X__ design specification
_X__ implementation
_X__ test derivation
____ RTL
____ netlists
____ transistor level
____ other: ____________
____

o Has the tool been integrated with other tools?
____ no
_X__ yes
with __SCR* toolset for tabular specifications_
with __PEP___________________
with ____________________
with ____________________
____ do not know

6. RESOURCES

o Resource requirements for the tool:
UNIX version __any standard UNIX, Linux________
Windows version __95/98, NT________
Mac version ___________
Memory: ____________

o Cost, rights and restrictions:
____ free, no license
_X__ free, license required
_X__ for educational and research use only
____ nominal distribution charge
____ fee for underlying tool(s)
____ flat license fee
____ per user license fee
____ royalties per use
____ other: ____________
____
____

o User background prerequisites (check all that apply):
_X__ BS degree
____ MS degree
____ Ph.D. degree
____ knowledge of logic
____ first-order
____ high order
_X__ familiarity with a high-level programming language
____ familiarity with process algebra

```
     _X__ familiarity with temporal logic
     ____ other: ____________
                 ____________
                 ____________
o User's learning curve, if all prerequisites are met:
     _X__ one month
     ____ two months
     ____ less than six months
     ____ other:
            _____ months
o Tool support
     _X__ upgrades/maintenance
           Last version produced at this date: __Spin 3.3.3, 1999__
     _X__ manual
           _X__ on the web
     ____ training
     ____ listserv
     ____ mailing list
     _X__ dedicated conference(s)/workshop(s)
           (annual, international, since 1995)
     ____ human "help line"
     _X__ book(s)
     _X__ journal publications
     _X__ other: __regular electronic newsletter
                   (mailed out and posted on the web page)_______
                 __proceedings of Spin workshops, on the web page__
                 __web page with source code extensions depository__
                 __bug reports and suggestions, to the newsletter_________
o Current contact.
     spin_list@research.bell-labs.com (newsletter)

7. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

o Verification mechanism(s) (check all that apply):
     ____ equivalence
     ____ modal logic
     _X__ temporal logic
            __LTL___
     _X__ system or process invariants
     _X__ other: __never claims (Buchi automata)__
                 __trace can be replayed in simulator to demonstrate
                   property violation__________
                 ____________
o Tool supports (check all that apply):
     _X__ optimization and state reduction mechanism
           using __partial order reduction,
                   bit-state hashing (optional),
                   Wolper's hash-compact method (optional),
                   storing reachable states with minimized automaton,
                   statement merging,
                   nested depth-first search algorithm____
     _X__ simulator
           _X__ interactive
           _X__ random
```

_X__ guided
_X__ feedback on in what state verification failed
X__ trace leading to the state
____ built-in support for checking for:
_X_ deadlock
_X_ livelock
_X_ boolean propositions
_X_ other: __LTL formulas (internally converted into never claims)
__dynamically growing and shrinking number of processes__
__semaphores__________

8. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

o Degree of proof mechanization.
____ fully mechanized
____ partially mechanized
o Support for developing and viewing the proof.
o Presentation of proof to the user (e.g., user input or canonical expressions, with or without quantifiers).
o Tool supports (check all that apply):
____ automated support for arithmetic reasoning
____ automated support for efficient handling of large propositional expressions
____ automated support for rewriting
____ possible to use lemmas before they are proved.
____ possible to state and use axioms without having to prove them.
____ new definitions can be introduced and existing definitions modified during proof
____ facilities for editing proofs
____ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified
____ reasonably easy to reverify a theorem after slight changes to the specification

9. OPEN-ENDED QUESTIONS

o Strengths of this tool.
Easy to learn by people with some programming experience.
Optimized for verifying large problem sizes (e.g. bit-state hashing, on-the-fly checking).
Actively contributing user community in more than 40 countries.
o Limitations of this tool.
Not efficient to specify large data sets.
o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.
Develop verified process control systems from requirements to implementation.
Trace logical design errors in distributed systems, such as operating systems, railway signaling protocols, data communications protocols, switching systems, concurrent algorithms.
o Applications that the tool was used for - case studies, examples,

success stories.

Posted throughout Spin News Letters and workshop proceedings, http://netlib.bell-labs.com/netlib/spin/news.

Some examples include: specification, design, verification and implementation of a safe object oriented process control application, verification of Java applications, steam boiler, hardware cache coherence protocols, NASA's fault tolerant embedded space craft controller, a multi-threaded plan execution programming language of NASA's New Millennium Remote Agent artificial intelligence based spacecraft control system architecture, telecommunications and security protocols, Dutch mobile sea-level control.

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1. http://eis.jpl.nasa.gov/quality/Formal_Methods/

NRL Protocol Analyzer

```
****************************************************************************
***************************** NRL PROTOCOL ANALYZER ******************
************************************ Sep. 1999************************
```

For this particular tool, please answer the following questions grouped based on: general description of the tool, tool implementation, tool features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

```
o Rough classification:
      _X__ model checker
      ____ theorem prover
      _X__ mechanized proof assistant
      ____ other: ________________
o Application domain(s) or class(es) of problems originally intended.
      Analysis of cryptographic protocols used to authenticate principals and
services
      and distribute keys in a network.
      Proving properties of security protocols and finding flaws in them.
o Intended audience.

o Language(s) and/or technique(s) that the tool is based on.
      NRL language, loosely resembling Prolog, used to
      model a protocol as a set of transitions of interacting state machines.
o Reasoning mechanisms used for the tool.
      Extended term-rewriting model of Dolev and Yao.
      Specify insecure states and prove them unreachable, by using either:
      exhaustive search backwards from the state; or
      proof techniques for reasoning about state models (using induction
      for infinite state and narrowing for word reduction).

o Comparable languages/tools.
      STeP.
```

2. TOOL IMPLEMENTATION

```
o Underlying mechanism of the tool's implementation.
      Prolog.
o How extensible and/or customizable is the tool.
      ____ source code given
      _X__ tool implemented in a public-domain language
      ____ other: ____________
                  ____________
                  ____________
```

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):

____ GUI
____ Library of standard types, functions, and other constructions
  ____ the library is validated

  The extent of the library is (speaking from the point of view of a potential user):
  ___ not very comprehensive
  ___ reasonably comprehensive
  ___ quite comprehensive

____ Editing and document preparation tools

  ____
  ____
  ____

____ Cross-referencing
____ Browsing
____ Requirements tracing
____ Incremental development across multiple sessions
____ Change control and version management
____ Consistency checking
____ Completeness checking
____ Other:

  ____
  ____
  ____
  ____
  ____
  ____
  ____
  ____

o How interactive/mechanized/automated is the tool.
  _X__ fully automated
  _X__ user guided
    __possible to switch between automated and manual mode__
  ____ other: ____________

4. TOOL INPUT AND OUTPUT

o Tool supports these models:
  ____ synchronous
  ____ asynchronous
  ____ mixed

o Input to the tool.
  Description of state in terms of words known by intruder and values of local state variables.
o Output from the tool.
  Complete description of all reachable states and non-redundant paths that may precede the specified state.
  Proof failed/passed.
o The language used for input to the tool has (check all that apply):
  _X__ formal semantics

____ modern programming language constructs (e.g. if-else):

____
____
____

____ strong typing
____ modularity
____ hierarchical design
_X__ parameterization
____ communication between processes
    ____ buffered
____ built-in model of computation
____ other: ____________
            ____
            ____

3. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):

____ requirements
_X__ design specification
____ implementation
____ test derivation
____ RTL
____ netlists
____ transistor level
____ other: ____________
            ____

o Has the tool been integrated with other tools?

____ no
____ yes
  with ____________________
  with ____________________
  with ____________________
____ do not know

Interface for a requirements language.
Interface for high-level security language CAPSL.

4. RESOURCES

o Resource requirements for the tool:

UNIX version __________
Windows version __________
Mac version __________
Memory: ____________

o Cost, rights and restrictions:

____ free, no license
____ free, license required
____ nominal distribution charge
____ fee for underlying tool(s)
____ free for educational and research use only

____ flat license fee
____ per user license fee
____ royalties per use
____ other: ____________
____
____

o User background prerequisites (check all that apply):

____ BS degree
____ MS degree
____ Ph.D. degree
_X__ knowledge of logic
    ____ first-order
    ____ high order
____ familiarity with a high-level programming language
____ familiarity with process algebra
____ familiarity with temporal logic
____ other: ____________
____
____

o User's learning curve, if all prerequisites are met:

____ one month
____ two months
____ less than six months
____ more than six months
    ______ months

o Tool support

_X__ upgrades/maintenance
    Last version produced at this date: __1999____

____ manual
    ____ on the web
____ training
____ listserv
____ mailing list
____ conference(s)/workshop(s)
____ human
____ book(s)
_X__ journal/conference publications
____ other: ____________
____

o Current contact.

Catherine Meadows
Code 5543, Naval Research Laboratory, Washington DC 20375
meadows@itd.nrl.navy.mil

http://www.itd.nrl.navy.mil/ITD/5540/projects/crypto.html

6. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

Note: we will consider the state exploration portion of NRL Protocol Analyzer as "model checker."

o Verification mechanism(s) (check all that apply):

_____ equivalence
_____ modal logic
_____ temporal logic
_____ system or process invariants
_____ built-in support for checking for:
    ___ deadlock
    ___ livelock
    ___ other: ____________
_____ other: __state exploration__________

o Tool supports (check all that apply):

_X__ optimization and state reduction mechanism
using __narrowing algorithm___
__built-in rules for discarding redundant/unreachable paths and states________________________
__user-generated rules using a database of formal languages____
_____ symbolic simulator:
    _____ interactive
    _____ random
_X__ feedback on in what state verification failed
    _X_ trace leading to the state

7. QUESTIONS ABOUT THEOREM PROVERS/MECHANIZED PROOF ASSISTANTS [NASA98]

Note: we will consider the proof-oriented part of NRL protocol Analyzer as "theorem prover".

o Degree of proof mechanization.

_____ fully mechanized
_____ partially mechanized

o Support for developing and viewing the proof.

o Presentation of proof to the user (e.g., user input or canonical expressions, with or without quantifiers).

o Tool supports (check all that apply):

_____ automated support for arithmetic reasoning
_____ automated support for efficient handling of large propositional expressions
_____ automated support for rewriting
_____ possible to use lemmas before they are proved.
_____ possible to state and use axioms without having to prove them.
_____ new definitions can be introduced and existing definitions modified during proof
_____ facilities for editing proofs
_____ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified

____ reasonably easy to reverify a theorem after slight changes to the specification

8. OPEN-ENDED QUESTIONS

o Capabilities of this tool.
o Limitations of this tool.
o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.
o Applications that the tool was used for - case studies, examples, success stories.

Questionnaire for potential users:
----

o Briefly describe problems that you need solved (in order to help us estimate if those problems can be addressed by formal tools).
o Have you used formal tools? If yes, for what application? What were the areas of satisfaction? What were the problem areas? What would you like to see in the future?
o Describe your dream toolkit.
o What would you consider a "good place" to integrate formal tools in existing or separate toolkits?

Questionnaire for tool makers/integrators:
----

o If you already produce and/or sell toolkits, would you be interested in integrating formal tools in the toolkit, and why.
o What information do you need in order to be able to integrate formal tools in a toolkit.

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1. http://eis.jpl.nasa.gov/quality/Formal_Methods/

*SCR**

```
**************************************************************************
*************** Software Cost Reduction (SCR*) ************************
*********************** Sep. 1999**************************************

For this particular tool, please answer the following questions grouped
based on: general description of the tool, tool implementation, tool
features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

o Rough classification:
      ____ model checker
      ____ theorem prover
      ____ mechanized proof assistant
      _X__ other:  __integrated environment.
                     Consistency checker and simulator
                     integrated with external tools:
                     model checker (Spin) and mechanized
                     proof assistant (PVS).
o Application domain(s) or class(es) of problems originally intended.
      Software requirements specification.
o Intended audience.
      Software developers.
o Language(s) and/or technique(s) that the tool is based on.
      SCR requirements method, based on tables.
o Reasoning mechanisms used for the tool.
      A form of classic state machine model.
o Comparable languages/tools.
      Requirements State Machine Language (RSML)/SMV, SVC.

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation.
      Currently: C, C++, executes on Sun workstations.
      New version, scheduled for October 1999, is implemented in Java
      and will execute on PC's.
o How extensible and/or customizable is the tool.
      ____ source code given
      _X__ tool implemented in a public-domain language
      _X__ other: __currently developing a toolset architecture that
                    will make the integration of external tools easier__
                  ____________

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):
      _X__ GUI
      ____ Library of standard types, functions, and other constructions
            ____ the library is validated
```

The extent of the library is (speaking from the point of view of a potential user):
___ not very comprehensive
___ reasonably comprehensive
___ quite comprehensive

_X__ Editing and document preparation tools
_specification editor for creating requirements specifications_____
____
____

_X__ Cross-referencing
_X__ dependency graph browser___

_X__ Browsing
____ Requirements tracing
_X__ Incremental development across multiple sessions
____ Change control and version management
_X__ Consistency checking
_X__ Completeness checking
____ Other:
__simulator, with visual front ends tailored to particular applications (e.g. cockpit controls)______________
__automatic derivation of more abstract models from SCR specifications (e.g. for more efficient model checking)_
__pretty-printer_____________
__typechecker_____________
__syntax checker_____________

o How interactive/mechanized/automated is the tool.
_X__ fully automated
_X__ user guided
____ other: ____________

4. TOOL INPUT AND OUTPUT

o Tool supports this kind of models:
____ synchronous
__X_ asynchronous
____ mixed

o Input to the tool.
Tabular SCR specification; asynchronous input from non-deterministic environment.

o Output from the tool.
Specification editor output:
__dictionaries with static information (e.g. names of variables, user-defined types)__
__tables__
Dependency graph browser:
__directed graph depicting dependencies among variables.__
Consistency checker:
__syntax and type errors, missing cases, variable name discrepancies, unwanted nondeterminism, and circular definitions.
Abstraction derivator:
__more abstract model, eliminated irrelevant variables and unneeded detail__

o The language used for input to the tool has (check all that apply):

_X__ formal semantics

____ modern programming language constructs (e.g. if-else):

____

____

____

_X__ strong typing

_X__ modularity

____ hierarchical design

____ parameterization

____ communication between processes

____ buffered

_X__ built-in model of computation

____ other: ____________

____

____

3. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):

_X__ requirements

____ design specification

____ implementation

____ test derivation

_x__ RTL

(under current investigation)

____ netlists

____ transistor level

_X__ other: __documentation______

__levels that can be addressed with Spin and PVS_

o Has the tool been integrated with other tools?

____ no

_X__ yes

with _Spin model checker__

with _PVS theorem prover__ using TAME high-level user interface

with ______________________

with ______________________

____ do not know

4. RESOURCES

o Resource requirements for the tool:

UNIX version ___SunOS____

Windows version ___for Oct'99 release___

Mac version ___________

Memory: _____________

o Cost, rights and restrictions:

____ free, no license

_X__ free, license required

_X__ for educational and research use only

____ nominal distribution charge

____ fee for underlying tool(s)

____ flat license fee

____ per user license fee

```
     ____ royalties per use
     ____ other: ____________
o User background prerequisites (check all that apply):
     _X__ BS degree
     ____ MS degree
     ____ Ph.D. degree
     ____ knowledge of logic
           ____ first-order
           ____ high order
     ____ familiarity with a high-level programming language
     ____ familiarity with process algebra
     ____ familiarity with temporal logic
     ____ other: ____________
                 ____________
                 ____________

o User's learning curve, if all prerequisites are met:
     _X__ one month
     ____ two months
     ____ less than six months
     ____ other
           ______ months
o Tool support
     _X__ upgrades/maintenance
           Last version produced at this date: _1998_____
     _X__ manual
           ____ on the web
     _X__ training
     ____ listserv
     ____ mailing list
     ____ dedicated conference(s)/workshop(s)
     ____ human "help line"
     ____ book(s)
     _X__ journal/conference publications
     ____ other: _________________
o Current contact.
     Naval Research Laboratory,
     Code 5546, Washington DC 20375
     kirby@itd.nrl.navy.mil
     labaw@itd.nrl.navy.mil

     http://www.chacs.itd.nrl.navy.mil/SCR

6. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

Note: this section applies to model checker Spin.

o Verification mechanism(s) (check all that apply):
     ____ equivalence
     ____ modal logic
     _X__ temporal logic
            __LTL___
     _X__ system or process invariants
     _X__ other: __never claims (Buchi automata)__
                 __trace can be replayed in simulator to demonstrate
```

property violation__________
____

o Tool supports (check all that apply):

_X__ optimization and state reduction mechanism
using __partial order reduction,
bit-state hashing (optional),
Wolper's hash-compact method (optional),
storing reachable states with minimized automaton,
statement merging,
nested depth-first search algorithm____

_X__ simulator
_X__ interactive
_X__ random
_X__ guided

_X__ feedback on in what state verification failed
X__ trace leading to the state

____ built-in support for checking for:
_X_ deadlock
_X_ livelock
_X_ boolean propositions
_X_ other: __LTL formulas (internally converted into never claims)
__dynamically growing and shrinking number of processes__
__semaphores__________

7. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

Note: this section applies to mechanized proof assistant PVS, with TAME interface and SCR validity checker.

o Degree of proof mechanization:
____ fully mechanized
_X__ partially mechanized
(although finite state verification and the proof of many straightforward results are fully automatic. There is also a batch mode in which proofs may be easily rerun, and a facility for defining proof strategies to automate proofs.

o Support for developing and viewing the proof:
Tcl/Tk interface to display proof trees and theory hierarchies.
Proofs yield scripts that may be edited, attached to additional formulas,
and rerun. Proofs may also be checkpointed, providing rapid access to parts of a proof the user wishes to examine or adjust.

o Presentation of proof to the user (e.g., user input or canonical expressions with or without quantifiers):
Proofs are presented in a sequent-style representation. PVS takes care to assure that the initial proof goal transparently reproduces the formula input by the user. Quantification is retained; implicit universal quantification in the user's specification is made explicit.

o Tool supports (check all that apply):
_X__ automated support for arithmetic reasoning
_X__ automated support for efficient handling of large propositional

expressions
_X__ automated support for rewriting
_X__ possible to use lemmas before they are proved.
_X__ possible to state and use axioms without having to prove them.
_X__ new definitions can be introduced and existing definitions modified during proof
_X__ facilities for editing proofs
_X__ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified
_X__ reasonably easy to reverify a theorem after slight changes to the specification
_X__ other:
  __integration with CTL model checking
  __ground evaluator (providing "run" speeds comparable to imperative programs)
  __proof strategies
  __proof storage, replay, and checkpointing
  __graphical display of proof trees, theory hierarchies, and prover commands
  __proof chain analysis
  __proof and theory status reporting

8. OPEN-ENDED QUESTIONS

o Capabilities of this tool.
  Mathematically founded tool for non-specialists in formal methods.
  Well-developed user interface.
o Limitations of this tool.
  Flat structure of specifications.
o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.
  Requirements specification, specification, verification, documentation.
o Applications that the tool was used for - case studies, examples, success stories.
  Listed in http://www.itd.nrl.navy.mil/ITD/5540/personnel/heitmeyer.html.
  Avionics systems, telephone networks, nuclear power plants, etc.:
  English-language requirements for NASA International Space Station.
  Requirements specification for flight guidance system.
  Specification and verification of contractor-developed: Weapons Control Panel, and a cryptographic system.

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1. http://eis.jpl.nasa.gov/quality/Formal_Methods/

Tatami

```
*********************************************************************
***************************** Tatami System ************************
******************************* Sep. 1999***************************
```

For this particular tool, please answer the following questions grouped based on: general description of the tool, tool implementation, tool features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

o Rough classification:
  ____ model checker
  ____ theorem prover
  _X__ mechanized proof assistant
  _X__ other: _integrated suite of tools: Kumo, web-based proof assistant; barista proof server; tatami database and protocol for data exchange; and truth maintenance system, for keeping track of users who are cooperating on the same proof. ________________

o Application domain(s) or class(es) of problems originally intended.
  Web-based cooperative design, specification and validation of software systems, especially concurrent OO systems.

o Intended audience.
  Software engineers.

o Language(s) and/or technique(s) that the tool is based on.
  OBJ3 (order sorted equational logic), BOBJ (extension of OBJ, first order logic with equations as atoms).

o Reasoning mechanisms used for the tool.
  Inference rules in first order logic with equational logic, including induction and coinduction.

o Comparable languages/tools.
  This system is an extension of CafeOBJ system, which is a network-based environment for supporting systematic creation, checking, verification and maintenance of OO formal specifications.

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation.
  Java 1.2, OBJ3.

o How extensible and/or customizable is the tool.
  ____ source code given
  _X__ tool implemented in a public-domain language

  ____ other: ____________
  ____
  ____

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):

_X__ GUI
____ Library of standard types, functions, and other constructions
____ the library is validated

The extent of the library is (speaking from the point of view of a potential user):
___ not very comprehensive
___ reasonably comprehensive
___ quite comprehensive

_X__ Editing and document preparation tools
____
____
____

____ Cross-referencing
_X__ Browsing
____ Requirements tracing
_X__ Incremental development across multiple sessions
_X__ Change control and version management
____ Consistency checking
____ Completeness checking
_X__ Other:
__executing proof scores on a remote server_____________
____
____
____
____
____
____
____

o How interactive/mechanized/automated is the tool.

_X__ fully automated
_X__ user guided
____ other: ____________

4. TOOL INPUT AND OUTPUT

o Tool supports these models:

____ synchronous
____ asynchronous
____ mixed

o Input to the tool.

Specification in BOBJ; prof script with execution commands in Duck language.

o Output from the tool.

Proof results.
Kumo generates web pages with documentation based on user input.

o The language used for input to the tool has (check all that apply):

_X__ formal semantics
_X__ modern programming language constructs (e.g. if-else):
____

____

____

____ strong typing
_X__ modularity
____ hierarchical design
_X__ parameterization
____ communication between processes
    ____ buffered
____ built-in model of computation
____ other: ____________

5. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):
_X__ requirements
_X__ design specification
_X__ implementation
____ test derivation
____ RTL
____ netlists
____ transistor level
____ other: ____________

o Has the tool been integrated with other tools?
____ no
_X__ yes - please name tool and applications
    with _CafeOBJ environment____________________
    with ____________________
    with ____________________
____ do not know

6. RESOURCES

o Resource requirements for the tool:
UNIX version __________
Windows version __________
Mac version __________
Memory: ____________

o Cost, rights and restrictions:
____ free, no license
____ free, license required
____ nominal distribution charge
____ fee for underlying tool(s)
____ free for educational and research use only
____ flat license fee
____ per user license fee
____ royalties per use
____ other: ____________

o User background prerequisites (check all that apply):

_X__ BS degree
____ MS degree
____ Ph.D. degree
____ knowledge of logic
  ____ first-order
  ____ high order
____ familiarity with a high-level programming language
____ familiarity with process algebra
____ familiarity with temporal logic
____ other: ____________
____
____

o User's learning curve, if all prerequisites are met:

____ one month
____ two months
_X__ less than six months
____ other
  ______ months

o Tool support

_X__ upgrades/maintenance
  Last version produced at this date: _1999_____

_X__ manual
  _X__ on the web
____ training
____ listserv
_X__ mailing list
  for CafeOBJ
_X__ dedicated conference(s)/workshop(s)
  for CafeOBJ
____ human "help line"
_X__ book(s)
  for OBJ3
_X__ journal/conference publications
____ other: ____________
____

o Current contact.

http://www-cse.ucsd.edu/groups/tatami/

7. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

o Verification mechanism(s) (check all that apply):

____ equivalence
____ modal logic
____ temporal logic
____ system or process invariants
____ built-in support for checking for:
  ___ deadlock
  ___ livelock
  ___ other: ____________
____ other: ____________

o Tool supports (check all that apply):

____ optimization and state reduction mechanism

using ______________________

____ symbolic simulator:

____ interactive

____ random

____ feedback on in what state verification failed

___ trace leading to the state

8. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

o Degree of proof mechanization.

____ fully mechanized

____ partially mechanized

o Support for developing and viewing the proof.

Web-based.

o Presentation of proof to the user (e.g., user input or canonical expressions,
with or without quantifiers).

o Tool supports (check all that apply):

____ automated support for arithmetic reasoning

____ automated support for efficient handling of large propositional expressions

____ automated support for rewriting

____ possible to use lemmas before they are proved.

____ possible to state and use axioms without having to prove them.

____ new definitions can be introduced and existing definitions modified during proof

____ facilities for editing proofs

____ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified

____ reasonably easy to reverify a theorem after slight changes to the specification

9. OPEN-ENDED QUESTIONS

o Capabilities of this tool.

Ease of use, user interface and system operation designed for software engineers who are not experts in formal methods.

Will be possible to use various proof checkers other than Kumo.

o Limitations of this tool.

Kumo is not a powerful proof assistant like HOL or PVS.

o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.

Cooperative web-based software system design and validation.

o Applications that the tool was used for - case studies, examples, success stories.

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1.

http://eis.jpl.nasa.gov/quality/Formal_Methods/

What Tool Makers Need for Tool Integration (1 received response)

```
Questionnaire for tool makers/integrators:
- -----------------------------------------
o If you already produce and/or sell toolkits, would you be interested
in integrating formal tools in the toolkit, and why.
      Integration is happening. Need a spectrum of tools for any kind
      of useful system.

o What information do you need in order to be able to integrate formal
tools in a toolkit.
      API, sockets main link into Z/EVES.
```

Appendix B: Formal Methods Term Taxonomy

Formal Methods Term Taxonomy

Background

Mature life-cycle process, in the context of system engineering, consists of: requirements definition, system design, high-level design, low-level design, implementation, testing (unit testing, component testing, and system testing), user support, and maintenance.

Model is a system of definitions, assumptions and equations, set up to represent and discuss physical phenomena and systems. In the context of mathematical logic, a model is an implementation, I, of a set of well-formed formulas of a formal language such that each member of the set is true in I.

Axiom is a mathematical formula that can assert arbitrary properties over arbitrary (new or existing) entities.

Definition, is an axiom that introduces a new symbol and gives its value or meaning as a function of previously existing symbols.

Theorem is a logical formula derived from axioms using inference rules.

Method, in the context in an engineering discipline, describes a way in which a process is to be conducted. In the context of system engineering, a method consists of: 1) underlying model of development; 2) a language or languages; 3) defined ordered steps; and 4) guidance for supplying them in a coherent manner.

Proof is a chain of reasoning using rules of inference and a set of axioms that leads to conclusion, i.e. it is derivation of a theorem.

Step-wise refinement, in the context of system engineering, is the process of deriving level $i+1$ of the process cycle from level i, and refining level i based on level $i+1$, in systematic fashion through all cycles of life-cycle.

Taxonomy

Abstraction is the process of simplifying and ignoring irrelevant details and focusing, distilling, and generalizing what remains. In formal methods, abstraction is a tool for eliminating

distracting detail, avoiding premature commitment to implementation choices, and focusing on the essence of the problem at hand.

Breadth-first search is a search that generates first all the immediate neighbors of a state, then all the next neighbors, and so on.

Completeness is a property defined as presence of all possible cases.

Consistency is a property defined as lack of conflicting cases.

Explicit model checking is a type of model checking in which the system to be analyzed is represented by enumerating its states and transitions. State exploration is performed over individual states. The term "model checking" usually implies explicit model checking.

Formal analysis is mathematically-based analysis.

Formal method is a mathematically-based technique for describing system components, properties and/or behavior. Formal methods are different than traditional engineering mathematics in the sense that they are used for describing digital systems, such as hardware and software, using logic and discrete mathematics. A formal method has an underlying theoretical model against which a description can be verified. It consists of a notation (i.e. formal specification language) and some form of deductive apparatus (i.e. proof system).

Formal methods may be applied at varying levels of rigor or formalization. Listed in order of increasing formality and effort, a suggestive guide to levels of rigor includes:

1. Use of notations and concepts derived from logic and discrete mathematics to develop more precise requirements statements and specifications. Proof, if any, is informal.
2. Use of formalized specification languages with mechanized support tools ranging from syntax checkers and prettyprinters to typecheckers.
3. Use of fully formal specification languages with rigorous semantics and correspondingly formal proof methods that support theorem proving and model checking.

Formal proof is a complete and mathematically based argument for the validity of a statement about a system description. A proof proceeds in a series of steps, each of which draws conclusions from a set of assumptions. Justification for each step is derived from a small set of rules which state what conclusions can be reasonably drawn from assumptions. Such justification eliminates ambiguity and subjectivity from the argument. Formal proofs may be prepared manually or, preferably, with the assistance of a formal methods tool.

Formal specification is a description of a planned or existing process, entity and/or system, written in a formal language. It is a concise and unambiguous description of the behavior and/or properties of the process/entity/system, and can be written at various levels of abstraction and formalization. It can be used for requirements, system design, high-level design, and low-level

design specification, as well as test derivation. The most formal specifications are written in languages with well-defined semantics that support formal deduction and allow the consequences of the specification to be calculated through proof of putative theorems.

Formal (specification) language is a mathematically based language, and has a formal syntax and semantics.

- Formal languages can be broadly classified as model-oriented, property-oriented, or a combination of both. Model-oriented languages explicitly model system behavior. Property-oriented language describe properties of the system.
- Formal languages can also be classified as sequential or concurrent, if they are used to specify sequential or concurrent systems, respectively. For example, process algebras are model-oriented languages which describe the behavior of concurrent systems by describing their algebra of communicating processes.
- Formal languages can be executable, and can have tool support.
- Programming languages are formal languages, but are not considered appropriate for use in formal specifications because of: insufficient abstraction ability (e.g. in "true" formal languages, types do not have to be directly implementable); often there is a lack of complete formal semantics.

Formal (methods) tool is a program that implements some aspect of formal analysis, thus providing mechanized, computer assisted support for formal analysis. Like formal methods, formal methods tools can be formalized to various levels of rigor, from syntax checkers to theorem provers.

Formal validation is a type of formal analysis in which an implementation is tested in execution to demonstrate that it satisfies its requirements specification. Informally, it is proving that the requirements are right, (i.e. we are building the desired system).

Formal verification is a form of formal analysis in which each level of development is proven to satisfy the requirements of its superior level, (i.e. formal specification satisfies the corresponding formal requirements specification, and implementation satisfies the corresponding formal specification). Informally, it is proving that a system is built to its requirements.

Formalization is the application of a certain level of mathematical rigor; or the act of formalizing an informal process, system or entity by making it more mathematically rigorous. In the context of using formal languages and tools, levels of formalization are (in increasing order):

1. Use of mathematical concepts and notation, informal analysis (if any), absence of mechanized assistance.
2. Use of formalized specification language with some mechanical support.
3. Use of formal specification language with comprehensive mechanized environment, which includes mechanized proof assistant/theorem prover and/or model checker.

Mechanized proof assistant is a formal tool that implements theorem proving in an interactive way, requiring the user to guide the proof steps.

Model checking is a type of formal analysis that relies on building a (usually finite) model of a system and checking that a desired property holds in that model. The verification task is to demonstrate that the system is a model that satisfies the putative property. The specification should be syntactically and semantically correct. The check is performed as an exhaustive or partial state space search, often breadth-first. Model checking is based on a verification algorithm and thus requires no assistance from the user, i.e. it is "automatic."

Model checker is a formal tool that implements model checking. Model checkers usually rely on various algorithms, such as bit-state hashing or symmetry, to reduce state space search, and/or in the case of very large systems could provide an option to perform nearly exhaustive state space search.

Theorem proving is a type of formal analysis in which a proof of a property is performed over a specification. Both the specification and its properties are expressed as formulas in some kind of mathematical logic. The verification task is to show that the formal specification of the system implies the formal statement of a putative system property. The specification should be syntactically and semantically correct.

Theorem prover is a formal tool that implements theorem proving in an automated way, not requiring user assistance.

Parser is a formal tool that checks syntactic consistency.

Requirements specification is a specification describing essential, necessary or desired attributes of a system or system components.

Rule of inference is a rule in mathematical logic that defines the reasoning that determines when a conclusion may be drawn from a set of premises. In a formal system, the rules of inference should guarantee that *if* the premises are true, *then* the conclusion is also true.

Specification animators (or **emulators**) are executable programs which reinterpret a formal specification into a high-level dynamically executable form. Specification animations are not formal in a strict sense, but support the formal requirements and design verification process by providing analysts with an early view of the high-level dynamic behavior of the requirements.

Symbolic execution is execution which does not require parameters to have known values, (i.e., allows parameters in symbolic form).

Symbolic model checking is an approach to model checking in which the system to be analyzed is described by equations or logical formulas. For example, a form of symbolic model checking uses the state reduction technique to analyze sets of states, represented as Boolean formulas,

instead of individual states. For illustration, let us consider the state in which V is set to 0. All states that have V set to 0 are marked, and all states that can reach the marked states in one step are marked. This procedure is repeated until no new states can be marked. This set of states is then analyzed.

Symbolic simulation is a form of simulation that allows input parameters to be supplied in symbolic form, (e.g. as variables or functions).

Traceability of requirements is a property which means that system-level requirements are traceable to identifiable (functional) subsystems, components, or interfaces.

Typechecking is a form of formal analysis that detects semantic inconsistencies and anomalies, ensuring that entities must match their declaration and be combined only with other entities of the same or compatible type.

Typechecker is a formal tool that implements typechecking.

Unparser (or pretty-printer) is a tool that translates internal representations into display, and outputs formatted text. Usually used at the specification level.

Questionnaire

Tools Makers/Users

```
*********************************************************************
*****************************  Tool name   ************************
***************************** current date  ************************
```

For this particular tool, please answer the following questions grouped based on: general description of the tool, tool implementation, tool features and utilities, applications and resources.

1. GENERAL DESCRIPTION OF THE TOOL

o Rough classification:

    ____ model checker
    ____ theorem prover
    ____ mechanized proof assistant
    ____ other: ________________

o Application domain(s) or class(es) of problems originally intended.

o Intended audience.

o Language(s) and/or technique(s) that the tool is based on.

o Reasoning mechanisms used for the tool.

o Comparable languages/tools.

2. TOOL IMPLEMENTATION

o Underlying mechanism of the tool's implementation.

o How extensible and/or customizable is the tool.

    ____ source code given
    ____ tool implemented in a public-domain language
    ____ not extensible by user

    ____ other: ____________
    ____
    ____

3. TOOL FEATURES AND UTILITIES

o Tool supports the following (check all that apply):

____ GUI

____ Library of standard types, functions, and other constructions

____ the library is validated

The extent of the library is (speaking from the point of view of a potential user):

___ not very comprehensive

___ reasonably comprehensive

___ quite comprehensive

____ Editing and document preparation tools

____

____

____

____ Cross-referencing

____ Browsing

____ Requirements tracing

____ Incremental development across multiple sessions

____ Change control and version management

____ Consistency checking

____ Completeness checking

____ Other:

____

____

____

____

____

____

____

____

o How interactive/mechanized/automated is the tool.

____ fully automated

____ user guided

____ other: ____________

4. TOOL INPUT AND OUTPUT

o Tool supports these models:
- ____ synchronous
- ____ asynchronous
- ____ mixed

o Input to the tool.

o Output from the tool.

o The language used for input to the tool has (check all that apply):
- ____ formal semantics
- ____ modern programming language constructs (e.g. if-else):
  - ____________
  - ____________
  - ____________
- ____ strong typing
- ____ modularity
- ____ hierarchical design
- ____ parameterization
- ____ communication between processes
  - ____ buffered
- ____ built-in model of computation
- ____ other: ____________
  - ____________
  - ____________

5. TOOL APPLICATION

o Abstraction level that the tool can address (check all that apply):
- ____ requirements
- ____ design specification
- ____ implementation
- ____ test derivation
- ____ RTL
- ____ netlists
- ____ transistor level
- ____ other: ____________
  - ____________

o Has the tool been integrated with other tools?
- ____ no
- ____ yes - please name tool and applications
  - with ____________________
  - with ____________________
  - with ____________________
- ____ do not know

6. RESOURCES

o Resource requirements for the tool:
  UNIX version ______
  Windows version ______
  Mac version ______
  Memory: ______

o Cost, rights and restrictions:
  ____ free, no license
  ____ free, license required
    ____ for educational and research use only
  ____ nominal distribution charge
  ____ fee for underlying tool(s)
  ____ flat license fee
  ____ per user license fee
  ____ royalties per use
  ____ other: ______
  ______
  ______

o User background prerequisites (check all that apply):
  ____ BS degree
  ____ MS degree
  ____ Ph.D. degree
  ____ knowledge of logic
    ____ first-order
    ____ high order
  ____ familiarity with a high-level programming language
  ____ familiarity with process algebra
  ____ familiarity with temporal logic
  ____ other: ______
  ______
  ______

o User's learning curve, if all prerequisites are met:
  ____ one month
  ____ two months
  ____ less than six months
  ____ more than six months
    ______ months

o Tool support
  ____ upgrades/maintenance
    Last version produced at this date: ______
  ____ manual
    ____ on the web
  ____ training
  ____ listserv
  ____ mailing list
  ____ dedicated conference(s)/workshop(s)
  ____ human "help line"
  ____ book(s)
  ____ journal/conference publications
  ____ other: ______

o Current contact. ______

7. QUESTIONS APPLYING TO MODEL CHECKERS ONLY

o Verification mechanism(s) (check all that apply):

____ equivalence
____ modal logic
____ temporal logic
____ system or process invariants
____ built-in support for checking for:
  ___ deadlock
  ___ livelock
  ___ other: ____________
  ____
  ____

____ other: ____________
____
____

o Tool supports (check all that apply):

____ optimization and state reduction mechanism using __________________________
____ simulator:
  ____ interactive
  ____ random
  ____ simbolic
____ feedback on in what state verification failed

  _____ trace leading to the state

____ other:
____
____

8. QUESTIONS ABOUT THEOREM PROVERS [NASA98]

o Degree of proof mechanization.

____ fully mechanized

____ partially mechanized

o Support for developing and viewing the proof.

o Presentation of proof to the user (e.g., user input or canonical expressions, with or without quantifiers).

o Tool supports (check all that apply):

____ automated support for arithmetic reasoning

____ automated support for efficient handling of large propositional expressions

____ automated support for rewriting

____ possible to use lemmas before they are proved.

____ possible to state and use axioms without having to prove them.

____ new definitions can be introduced and existing definitions modified during proof

____ facilities for editing proofs

____ the foundations (i.e., all axioms, definitions, assumptions, lemmas) of the proof are identified

____ reasonably easy to reverify a theorem after slight changes to the specification

9. OPEN-ENDED QUESTIONS

o Capabilities of this tool.

o Limitations of this tool.

o Estimated possible uses of the tool, such as applications, classes of problems, stages of production cycle.

o Applications that the tool was used for - case studies, examples, success stories.

References:

[NASA98] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems", vol.1. http://eis.jpl.nasa.gov/quality/Formal_Methods/

Questionnaire

Potential Users

- Briefly describe problems that you need solved (in order to help us estimate if those problems can be addressed by formal tools)
- Have you used formal tools? If yes,
    - For what application?
    - What were the areas of satisfaction?
    - What were the problem areas?
    - What would you like to see in the future?
- Describe your dream toolkit.
- What would you consider a "good place" to integrate formal tools in existing or separate toolkits?

Questionnaire

Tools Makers/Integrators

- If you already produce and/or sell toolkits, would you be interested in integrating formal tools into the toolkit, and why?

- What information do you need in order to be able to integrate formal tools into a toolkit?

☆U.S. GOVERNMENT PRINTING OFFICE: 2002-710-038-10234

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of Information Systems Science and Technology to meet Air Force unique requirements for Information Dominance and its transition to aerospace systems to meet Air Force needs.